#121 NOVEMBER 1986

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

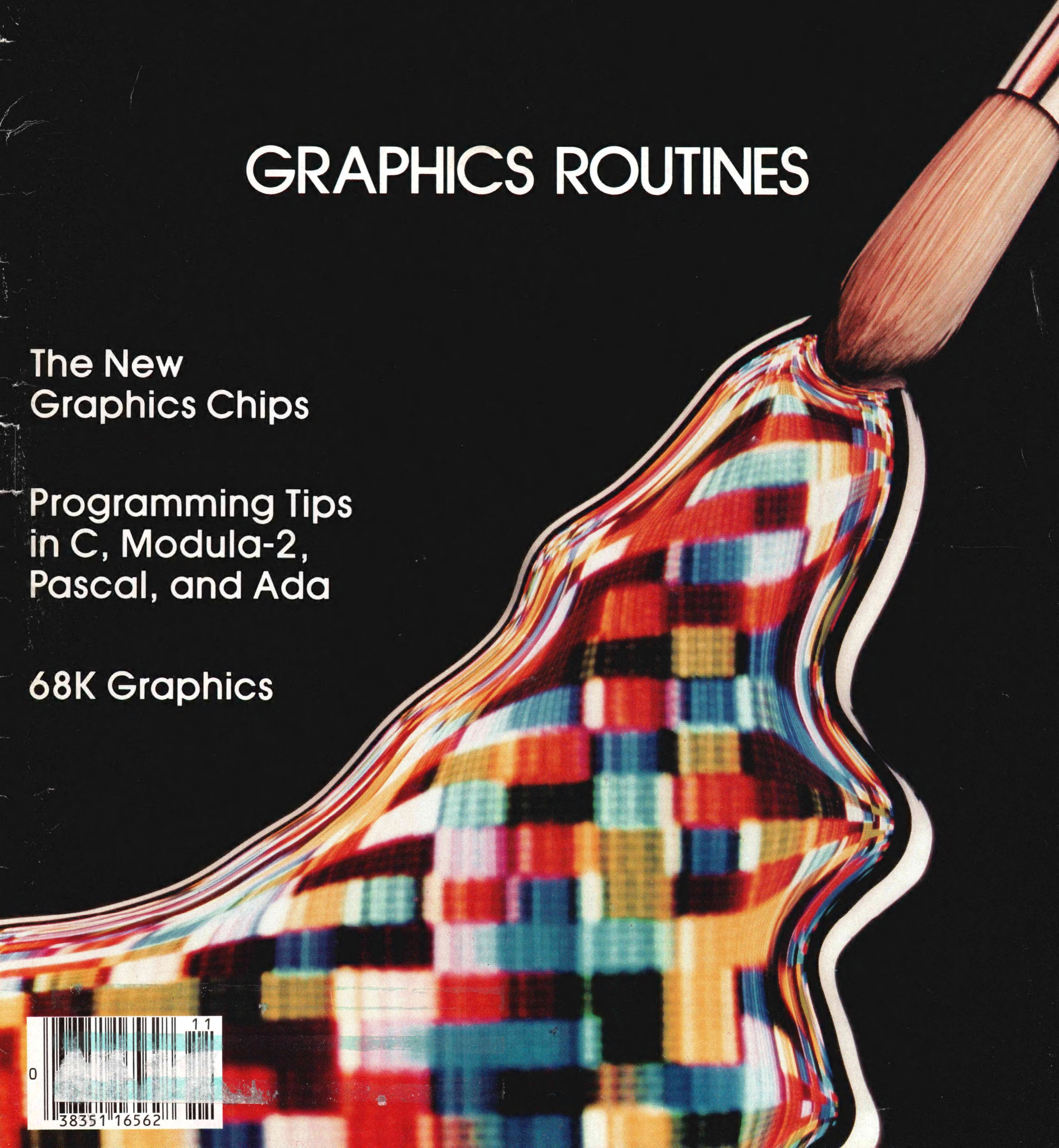
FOR THE PROFESSIONAL PROGRAMMER

GRAPHICS ROUTINES

The New Graphics Chips

Programming Tips in C, Modula-2, Pascal, and Ada

68K Graphics



0 38351 16562 11


Clipper is the fastest dBase III and dBase III Plus™ compiler available. Nothing else comes close. When performance counts, experts rely on Clipper for more speed, more power, and more creative freedom. You can, too. Call for details.
• Clipper compiled programs run 2 to 20 times faster.
• No royalties...no runtime fees.
• Source code security.
• User defined functions.
• Arrays.
• Simple menu commands.
• Context sensitive help can be included with programs.
• More fields; more memory variables.
• Call C and Assembly programs.
• Complete debugging facilities.
• Multiple file relationships.
• IBM PC, XT, AT, 3270 compatible™.
• Multi-user capability.
Clipper™
CLIPPER. THE dBASE COMPILER.
A WINNING PERFORMANCE EVERY TIME.
nantucket™
Nantucket Corporation
5995 South Sepulveda Boulevard
Culver City, California 90230
(213) 390-7923
Outside California call toll-free:
1-800-251-8438
dBase, dBase III, and dBase III Plus are trademarks of Ashton-Tate, Inc.
IBM PC, XT, AT, and 3270 are trademarks of International Business Machines Corporation.
Clipper and Nantucket are trademarks of Nantucket Corporation.
Circle no. 220 on reader service card.

MARK WILLIAMS C. AN ENLIGHTENING DEVELOPMENT FOR ATARI ST USERS.

If you've tried your hand at developing applications on the Atari ST, you know the problem. Programming tools aren't only hard to come by, they're hard to use. One might even say primitive. But now for some enlightening news: you can have all the power, portability and versatility of the C language from a leader in professional C programming tools, Mark Williams.

BRING YOUR PROGRAMMING UP TO SPEED.

The Mark Williams C compiler produces fast, dense code and supports the complete Kernighan & Ritchie industry standard C. You'll have access to GEM's AES and VDI libraries for programs using graphics, icons and the Atari mouse. And Mark Williams C lets you take advantage of the full 16 megabytes in Atari's 68000 microprocessor.

STREAMLINE DEVELOPMENT WITH POWER UTILITIES.

Mark Williams C is loaded with everything you'll need for professional development. Bring the power of the UNIX environment to your Atari ST with our Microshell Command Processor including pipes, I/O redirection and more. Edit your program with the highly acclaimed MicroEMACS full screen editor. Accelerate and simplify compiling with *make* which finds and recompiles only those modules affected by your changes. Then, when you're ready for debugging, call on our db Symbolic Debugger with single step, breakpoint and stack traceback functions. Over 40 commands, including a linker and assembler, provide a total development package for your Atari ST.

Features

C compiler
- Complete Kernighan & Ritchie C plus extensions
- Up to eight register variables
- Full access to AES and VDI libraries for programs using graphics, icons and mouse
- Complete UNIX-compatible libraries allow easy portability to and from UNIX development environment.
- Over 300 Atari-specific routines
- One-step compiling, linking with cc command
- English error messages
- Lint-like error checking

Microshell Command Processor, powerful UNIX style shell includes I/O redirection, pipes, command substitutions

MicroEMACS Full Screen Editor with commented source code included

Make Program Building Discipline

Complete symbolic debugger with single-step, breakpoints and stack traceback

Assembler, linker and archiver

Powerful Utilities Package: egrep, sort, diff, cmp, pr, tail, uniq, wc and more

Over 600 pages of documentation including 120 sample C programs

Not copy protected

MARK WILLIAMS C FOR THE ATARI ST
$179.95
60 DAY MONEY BACK GUARANTEE

DEPEND ON A NAME WITH A HISTORY OF PERFORMANCE.

Mark Williams C for the Atari ST is part of our growing line of C compilers. A line that includes the C compiler chosen by DEC, Intel, Wang and thousands of professional programmers. Now our Atari C compiler is earning its own reputation:

"Finally a great C compiler that exploits the power of the ST"–Sigmund Hartmann, President, Atari Software Group

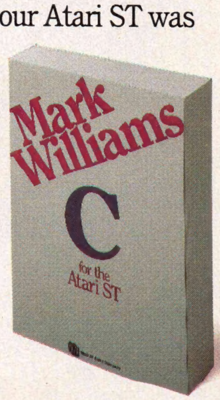
"The all-around best choice for serious software development on the ST."–Douglas Weir of *ANALOG COMPUTING*

GET WHAT YOUR ATARI ST HAS BEEN WAITING FOR.

Mark Williams C is just what your Atari ST was made for: powerful, professional programming. So now that you can have Mark Williams C for just $179.95, what are you waiting for?

Ask your Atari dealer about Mark Williams C or order today by calling **1-800-MWC-1700.***

*In Illinois call: 312-472-6659



Mark Williams Company
1430 West Wrightwood, Chicago, Illinois 60614

Circle **no. 102** on reader service card.


Instant Replay
NEW
Synthetic Intelligence
Patent Pending
Generates:
DEMOS, TUTORIALS, PROTOTYPES, PRESENTATIONS, TIMED KEYBOARD MACROS, AND MENU SYSTEMS
Instant Replay memorizes how you ran your program and Instantly generates a Demo Replay. Keystrokes and pause times, inserted prompts, pop-ups and prototypes are all memorized.
Screen Painter for creating 100% user designable pop-ups and Menu Windows. "Excellent, intuitive, easiest-to-use screen generator reviewed" PC Magazine .
Run-Time Screen Scanner and File Scanner for creating Menu systems and presentations.
Screen Grabber so you can edit and include any screen.
Text Editor - fast, full function, with pull-down menus that can be tailored. Memorize it and distribute it as a Demo!
200 Page Manual, 4 diskettes, 60 Day money back guarantee.
(Not Copy Protected)
Call or Write. We accept Visa, Amex, Master Card, COD, PO.
Dealer Inquiries Welcome.
Instant Replay(tm) $ 89.95
Demo Diskette $ 5.00
Dealer Poster $ 3.00
Nostradamus
Nostradamus Inc. / 5320 South 900 East, Suite 110
SLC, Utah 84117 / Order by phone (801) 261-0769
For IBM PC / XT / AT and True Compatibles
Ask about Programming Libraries
Circle no. 251 on reader service card.

CONTENTS

VOLUME 11, ISSUE 11

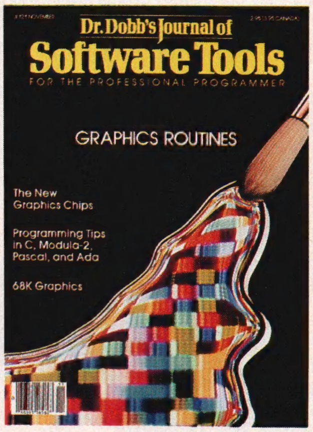
Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

ARTICLES

COLUMNS

FORUM

PROGRAMMER'S SERVICES



About the Cover

Nick Turner generated the color pixels on a plain old Apple II; Michael Hollister and photographer Michael Carr transferred them to the glistening surface of a half-pound of liquid mercury to simultaneously symbolize and achieve the blending of technology and art.

This Issue

The release of the Apple IIGS this fall (see Nick's brief report in Running Light, page 8) is sparking consumer interest in computer graphics, but to programmers the arrival of graphics chips that can provide up to 10,000-fold speed increases may be even more intriguing. Our lead article shows how to use the new chips from Intel and TI in your programs. We also present a digital dissolve routine that selects the most appropriate from among three algorithms, and a clever Macintosh graphics application in 68000 assembly language.

Next Issue

December is Operating Systems month at *DDJ*. The new 80386-based computers have got us intrigued with multitasking, and next month we'll publish the task scheduler component of a multitasking operating system for PC-class machines.

LANGUAGES THAT ARE CAUSING THE BIGGEST PROGRAMMING BACKLOG IN HISTORY ARE ALSO EATING NICE BIG HOLES IN OUR POCKETS.

Whether it's BASIC, COBOL, Pascal, "C", or a data base manager, you're being held back.

Held back because the language has frustrating limitations, and the programming environment isn't intuitive enough to keep track of what you're working on.

In the real world, there's pressure to do more impressive work, in less time, and for more clients.

We've been given some incredibly powerful hardware in recent times, but the languages aren't a whole lot better than they were 20 years ago.

So, whatever language you have chosen, by now you feel it's out to get you — because it is.

Sure, no language is perfect, but you have to wonder, "Am I getting all I deserve?"

And, like money, you'll never have enough.

Pretty dismal, huh?

We thought so, too.

So we did something about it.

We call it CLARION.™

You'll call it *"incredible."*

Distributed on 7 diskettes, CLARION consists of over 200,000 lines of code, taking 3+ years to hone to "world-class" performance.

With CLARION you can write, compile, run and debug complex applications in a New York afternoon.

Even if you're in Savannah.

It gives you the power and speed to create screens, windows and reports of such richness and clarity you would never attempt them with any other language.

Because *you* would have to write the code.

With CLARION you simply design the screens using our SCREENER utility and then CLARION writes the source code AND compiles it for you in seconds.

Likewise, you can use REPORTER to create reports.

Remember, only CLARION can recompile and display a screen or report layout for modification.

And with no time wasted.

All the power and facilities you need to write great programs, faster than you ever dreamed of.

Programs that are easy to use.

Programs that are a pleasure to write.

And to you that means true satisfaction.

You've coveted those nifty pop-up help windows some major applications feature. But you can't afford the time and energy it takes to write them into your programs.

That's the way it used to be.

So we fixed that, too.

CLARION's HELPER is an interactive utility that let's you design the most effective pop-up help screens that you can imagine. And they're "context sensitive," meaning you can have help for every field in your application.

Unlike the other micro languages, CLARION provides declarations, procedures, and functions to process dates, strings, screens, reports, indexed files, DOS files and memory tables.

Imagine making source program changes with the CLARION EDITOR. A single keystroke terminates the EDITOR, loads the COMPILER, compiles the program, loads the PROCESSOR and executes the program. It's that easy!

Our data management capabilities are phenomenal. CLARION files permit any number of composite keys which are updated dynamically.

A file may have as many keys as it needs. Each key may be composed of any fields in any order. And key files are updated whenever the value of the key changes.

Like SCREENER and REPORTER, CLARION's FILER utility also has a piece of the CLARION COMPILER. To create a new file, you name the Source Module. Then you name the Statement Label of a file structure within it.

FILER will also automatically rebuild existing files to match a changed file structure. It creates a new record for every existing record, copying the existing fields and initializing new ones.

Sounds pretty complicated, huh? Not with CLARION's documentation and on-line help screens. If you are currently competent in BASIC, Pascal or "C" you can be writing CLARION applications in a day. In two days you won't believe the eloquence of your CLARION programs.

Okay, now for the best part of all. You can say it in CLARION for $295.00—plus shipping and handling. All you need is an IBM® PC, XT, AT or true compatible, with 320 KB of memory, a hard disk drive, and a parallel port. And we'll allow a full 30 day evaluation period. If you're not satisfied with CLARION, simply return it in its original condition for a full refund.

If you're not quite ready to take advantage of this no-risk opportunity, ask for our detailed 16 page color brochure. It vividly illustrates the elegance of CLARION. Consider it a preview of programming in the fast lane.

Either way, the 800 call's a freebie.

SAY IT IN

CLARION™

A4ST/6

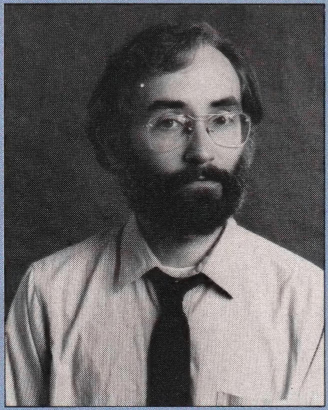
1-800-354-5444

VISA · AMERICAN EXPRESS · MasterCard

BARRINGTON SYSTEMS, INC. 150 EAST SAMPLE ROAD POMPANO BEACH, FLORIDA 33064 305/785-4555

Circle **no. 115** on reader service card.

EDITORIAL



Jim Anderson, the president of Digitalk, wrote to say that the classification system of his Smalltalk/V solves a problem posed in my Swaine's Flames column in August. Specifically, he says, it allows the programmer to handle physical units such as temperature and humidity realistically. He's right, but the scheme I was passing along in that column, and it's not a new idea, was to build into a compiler the constraints and checks that people working with physical quantities have to deal with. Things such as not being able to add apples and oranges, having feet times feet yield square feet, and having the units in a complex calculation multiply and divide and cancel out independently of (or in parallel with?) the quantities.

Built into the compiler, such features would constrain the programmer even more than the strong data typing of Modula-2. Laid on top of data-type constraints would be constraints that allowed multiplying two quantities but not adding them, and perhaps automatic rescaling that would take some control of numeric precision out of the programmer's hands. Why consider such constraints? The benefits would seem to be increased programmer productivity and increased program maintainability. The costs would be in generality and programmer control. These sound like the trade-offs of fourth-generation-language design.

Most 4GLs are specialized languages, appropriate in only a limited domain. They at least discourage, if not prohibit, low-level access to machines. They are designed to produce easily maintained code quickly.

If we believe Prentice-Hall's human book machine, James Martin, most 4GLs are remarkably unknown among those who know the most about programming. Martin recently polled "experts on programming languages" (including, I assume, professional software developers) and found few who even recognized the names of the most powerful fourth-generation languages. Why?

If fourth-generation languages are perceived as being bad languages in some sense—if, for example, they are seen as imposing too great a performance cost—maybe that explains their being ignored. Unfortunately, their being ignored means in turn that if the perception is wrong it will get corrected only slowly and if it is right the 4GLs' problems will get corrected slowly, because they won't benefit from the useful feedback of those experts who understand their faults.

There are signs that fourth-generation languages may have been changing while unobserved and may deserve another look. Fourth-generation languages are no longer restricted to mainframes and minicomputers. Vendors of fourth-generation languages for micros acknowledge some performance limitations of past 4GLs but claim that today's products produce fast, efficient code. They claim that any perception that fourth-generation languages are unworthy of the attention of serious software developers, if it was ever correct, is no longer.

Perhaps we should all take a look at this claim and at modern fourth-generation languages. At the very least, we should recognize that 4GLs need not be just elaborated database managers.

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of
Software Tools

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing, Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submissions: Send manuscripts and disk (with article and listings) to the Assistant Editor.

Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

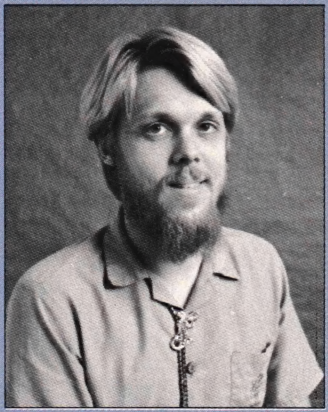
Customer Service: For subscription problems call: outside CA (800) 321-3333; within CA (619) 485-9623 or 566-6947. For order problems call (415) 366-3600.

Subscription Rates: $29.97 per year, U.S. Foreign rates: $63.97, air; $51.97, surface. Foreign subscriptions must be prepaid in U.S. dollars, drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service, Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

People's Computer Company
Dr. Dobb's Journal of Software Tools is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

RUNNING LIGHT



Apple's new IIGS, the high-end Apple II system, has just been introduced. The first batch of these platinum-cased beauties, with their 65816 processors and 256K memories, have even been signed by Steve Wozniak himself. "GS" stands for graphics and sound, and the IIGS lives up to the title grandly. It's got everything I've ever wanted from a II-series computer and much, much more. It's about time Apple did something to really spark the II line. Now the ball is in our court; we're the software designers, and it's our efforts that will directly influence the success or failure of this new machine.

In every issue of *DDJ*, toward the middle of the magazine, there's an insert with some tear-out cards. The top two cards are subscription cards. (You don't have a subscription? You know what to do.) The third card is very special: it's where you have a chance to talk directly to us. Your comments on that card are collected every month into a stack. The problem is, each month the stack is only about an inch high. We would love to see that stack grow to a medium-size mound, or even a small hill. So get out your pen or pencil and rip out that little card. We want to know what you think of the magazine, and we promise to read all the cards.

We have some really interesting themes coming up in future issues. Our May issue, for example, will focus on computer music. Who makes the best computer music? What machines are the most musical? Is MIDI the only way to interface? What's hottest and newest? We've never done a music issue before, so we especially need to hear from authors and experts as soon as possible. Call me at (415) 366-3600 if you have any ideas.

In June we'll be looking at telecommunications. A lot has happened since our last telecom issue. If you're working on something new and interesting, perhaps you have an article to write for us? We're especially interested in material about high-speed communications. Is there anybody who would like to write about fiber optics? What about a piece on how to program a signal processor CPU? Anything new in data compression or error detection/correction? Call me with ideas.

Our July issue, once again, will be full of Forth. We encourage all you dedicated and persistent Forthians to take part as you have in the past. Recently we've been able to increase our coverage of Forth—Michael Ham's contributions in particular have added a special flavor to the magazine (thank you, Mike!). Let's put together a really great seventh annual Forth issue.

August will be our C issue, and September will deal with algorithms. We're also thinking about covering fourth-generation languages in September. What do you think? Should we cover them? Would you like to write an article?

All in all, 1987 promises to be a banner year for *DDJ*. We're consistently receiving more high-quality article submissions than ever before. Your voices are being heard, both in the Letters column and in your direct phone calls to me. Let's keep improving together.

Nick Turner

Nick Turner
editor

ARCHIVES

DDJ Looks at the IBM PC

"On August 12, IBM announced the IBM Personal Computer, a small home machine. It will be sold . . . through Computerland stores and Sears Business Machines outlets.

"The prices quoted for the machine ranges from $1600 for a 16K starter system up to $6300 for a machine with two disks and 256K of storage. . . . we're exited about this announcement. IBM's presence in the market will intensify competition; the standards of support and documentaion will go up and the prices will go down as the other makers prepare to meet the challenge."—*Dave Cortesi*, DDJ, *October 1981.*

"We've had our hands on an IBM Personal Computer for a few days now. It's just a computer, after all."—*Dave Cortesi*, DDJ, *June 1982.*

CP/M-80 was a sturdy little tyke, but by no means was it a complete operating system. After examining these descendants of CP/M-80, my strongest feeling is one of disappointment. It disappoints me that two high-powered software houses have worked so hard to produce only . . . two more CP/Ms. —CP/M vs. MSDOS: A Technical Comparison, *Dave Cortesi*, DDJ, *July, 1982.*

DDJ Looks at its Readers

"We were amazed at how rapidly many of our readers have aquired IBM's Personal Computer. Equally interesting was that the 68000 chip has a greater following among our readers than the 8606/8088, although the margin is not overwhelming. Among topics eliciting very favorable response were CP/M, algorithms, assembly language, compilers and the Z80/8080. Pascal and Small-C/C were among the languages of evident enthusiasm, and multi-user systems are becoming worthy of more editorial attention."—*Marlin Ouverson*, DDJ, *August 1982.*

Ten Years Ago in DDJ

DDJ SEEKS SUPER LOGO! Like all massive organizations intent upon changing the fabric of society, Dr. Dobb's Journal has concluded that it should have a logo—a symbol by which all people may instanly recognize us. It might be our current title masthead...but that's SO longwinded. Ideally, it should be a symbol or figure that in some sense illustrates our activities (now, now—be nice)."—*DDJ, November/December 1976.*

"The Poly-88 system—which has replaced my Altair 8800 and my IMSAI 8080—has but two controls on the box. An on/off switch with a power indicator light, and a reset button with a halt light. That's all you get; that's all you need. It surely doesn't look impressive. Sort of like a toaster . . ."—*Jef Raskin*, DDJ, *November/December 1976.*

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

Large model C compiler performs like big-name brands includes free C tutorial

"As good as or better than most of the heavyweights. . ."
DR. DOBB'S, August 1986

The DATALIGHT C COMPILER is a full-implementation of the C language as defined in *The C Programming Language* by Kernighan and Ritchie. Supporting five memory models, DATALIGHT C has very fast compile, link, and execution times with a minimum of memory required. Our special introductory price is only $99 for the DEVELOPER'S KIT.

Optimize Your Code Generation

Now you can produce highly optimized code in the standard Intel object module format. The optimizations performed include common subexpression elimination, branch optimizations, constant folding, strength reductions, dead-code eliminations, and switch table compaction.

Five Memory Models Supported

DATALIGHT C provides five memory models so you can use the model that best suits your application.

Memory Models

Model	Code	Data
Compact	64k	total code & data
Small	64k	64k
Program	1M	64k
Data	64k	1M
Large	1M	1M

Compiling, One step...

Now with the one step DLC program you can compile, link, and create a .COM file. Also, one or more files can be compiled and linked using DLC.

Complete Library Includes Source Code

The UNIX compatible library includes complete source code. Experienced programmers can use the source code to configure and rebuild the library to suit the application.

Concise Documentation Included

The DATALIGHT C COMPILER and DEVELOPER'S KIT include a concise, to the point, programmer's manual. The 210-page manual is contained in an IBM-style three-ring binder which includes nine chapters, appendices, index, and easy to follow examples.

30-DAY MONEY-BACK GUARANTEE

Try our DEVELOPER'S KIT for 30 days, and if you do not find it to be equal to, or better than, any compiler you have, or are using, then you may return it for a full refund. Also, if you return the compiler within 30 days, you may keep the C tutorial, a $39 value, for trying our DEVELOPER'S KIT.

Introductory Prices

DATALIGHT C without source $60
(compact & small memory model support)

DEVELOPER'S KIT with source $99
(five memory models supported)

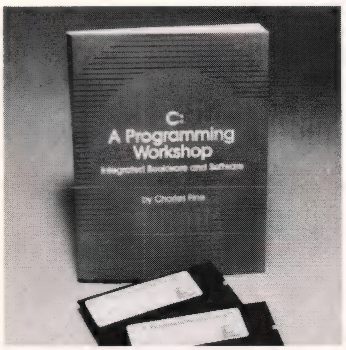
Not Copy Protected

ORDER TOLL-FREE TODAY!

1-(800) 221-6630

*SPECIAL OFFER!

ORDER the DEVELOPER'S KIT by December 31, 1986 and receive FREE...



"Combines print and software technology to create the most integrated new type of training system I have ever seen."

ADAM GREEN, INFO WORLD
January 27, 1986

"C:A Programming Workshop" is a complete C language learning package with an integrated compiler and screen editor. Learn the C language with tutorials, quizzes, and program exercises.

Add $5 for shipping in US/$15 outside US
COD (add $2.50)

*Limited offer available exclusively to readers who purchase directly from DATALIGHT.

Magazine Reviewers Shocked by DATALIGHT'S PERFORMANCE...

"Reviewing this compiler was quite a surprise for us. For such a low price, we were expecting a "lightweight" compiler. What we got was a package that is as good as or better than most of the "heavyweights." DATALIGHT C implements a complete C language. It also compiles quickly, doesn't take up much disk space, and looks impressive in the benchmarks."

DR. DOBB'S, August 1986

"This is a sharp compiler! ...what is impressive is that DATALIGHT not only stole the compile time show completely, but had the fastest Fibonacci executable time and had excellent object file sizes to boot!"

Chris Skelly, COMPUTER LANGUAGE
February 1986

DEVELOPER'S KIT (VERSION 2.12)

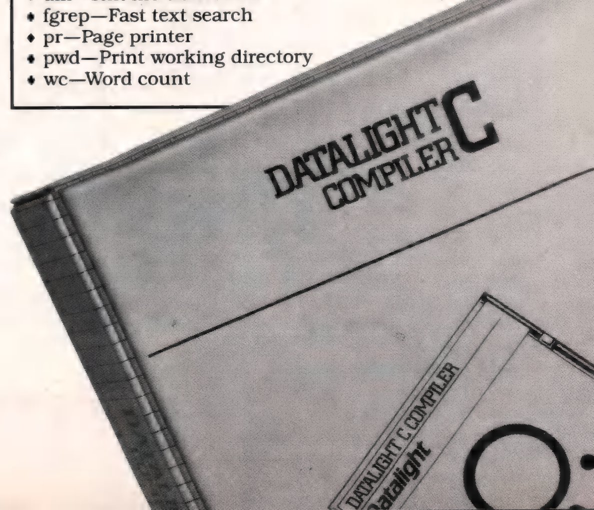
- Full UNIX System 5 C language plus ANSI extensions.
- Fast/tight code via powerful optimizations including common sub-expression elimination.
- DLC one step compile/link program.
- Multiple memory model support.
- UNIX compatible library with PC functions.
- Compatible with DOS linker and assembler.
- Third-party library support.
- Automatic generation of .COM files.
- Supports DOS pathnames, wild cards, and Input/Output redirection.
- Compatible with Lattice C version 2.x.
- Interrupt handling in C.
- Debugger support.
- ROMable code support/start-up source.

MAKE Maintenance Utility

- Macro definition support.
- MS-DOS internal commands.
- Inference rule support.
- TOUCH date manager.

Tools in Source Code

- cat—UNIX style "type"
- diff—Text file differences
- fgrep—Fast text search
- pr—Page printer
- pwd—Print working directory
- wc—Word count



Datalight

BOX 82441
KENMORE, WA 98028
(206) 367-1803

Circle **no. 203** on reader service card.

LETTERS

July Forth

Dear *DDJ*,

Thank you for the Forth issue, with some interesting and thought-provoking essays. I was even more gratified by the promise of more to come. I am sure you realize the vacuum we of the Forth community are discovering ourselves in. The language is far from dead, but meaningful forums appear to be on the endangered list.

Thank you for Mike Ham's column. Mike is one of the truly erudite spokesmen for Forth. It is a pleasure to see he has a platform as respected as *DDJ* from which to share his views.

I might add I have enjoyed comments from both Mike and Ray Duncan on the DDJ Forum on CompuServe. I look forward to seeing their insight on future pages of *DDJ*.

Again, thank you.

Gary Smith
P.O. Drawer 7680
Little Rock, AR 72217

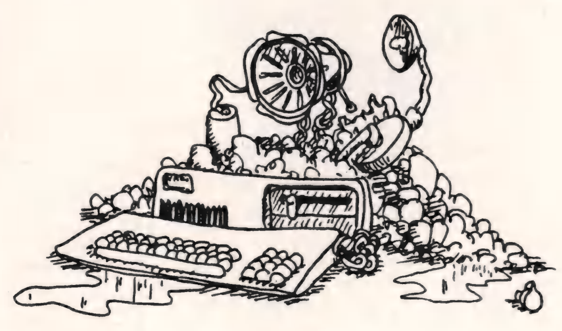
Dear *DDJ*,

The first sentence of George W. Shaw II's "Extended Control Structures" (July 1986), "The control structures in the Forth 83 Standard leave something to be desired," is an opinion that I stated three years ago in my paper referenced by Shaw. Since then I have realized that it was not the control structures that left something to be desired but my knowledge of how to use them to write clear and understandable programs. The standard control structures are just fine, and new structures are unnecessary.

Shaw is making the same mistake that I made three years ago. Because Forth does not have all the control structures that other structured languages have, I thought something was missing. But Forth does not need other control structure words.

Shaw's examples of extended logical structures are all instances of unsimple logic. He acknowledges that they are useful only 10 percent of the time. In Forth, when the logic becomes the least bit complicated, it is time to factor the complication out and give it a name telling what it does (but not how it does it). This is also a good idea in other languages. If this is done to Shaw's examples involving *BEGIN* and *IF*, his *LEAVES* can simply become *EXIT THEN*.

In traditional Forth systems, the same factoring cannot be done with examples involving *DO*. The self-styled Forth 83 Standard says that *EXIT* "may not be used within a *DO* loop." Rather than introduce new, novel, little-used, and hard-to-teach logic structures, let's stick to the structures that are already provided but remove the above-mentioned clause and allow *EXIT* to be used anywhere in a definition. This is easy to implement—either check compiler security or increment a counter for each *DO*, decrement it for each *LOOP* or *+LOOP*, and make *EXIT* smart enough to know how many loops to undo. It will also remove a restriction in the language and make Forth easier to learn and use. The awkwardness that Shaw ascribes to handling his tithe of logic disappears by removing narrow restrictions imposed on existing words, not on defining new words.



DIGITAL DISSOLVE effects ON Bit mapped SCREENS

Without giving any justification, Shaw asserts that the new Forth 83 *DO* loop is better than the Forth 79 *DO* loop. There are many who disagree. I do agree that the immediate *LEAVE* is better—it makes it convenient to observe the restrictions of structured programming. I also agree that *LEAVE* should work with *BEGIN* as well as with *DO*. But Shaw's argument that *LEAVE* is always followed by *THEN* and so should have a variation that implies *THEN* also applies to *EXIT*, *QUIT*, and *ABORT*. Rather than select one, some, or all of them for special consideration, leave well enough alone. The invariable *THEN* aids the reader to see the overall logical structure in a definition.

The natural meaning of *LEAVES*, found in some Forths, is that *2 LEAVES* escapes two loops, *3 LEAVES* escapes three loops, and so on.

Shaw's scheme for extending *IF* is intended to be general, but it does not allow nesting.

Wil Baden
339 Princeton Dr.
Costa Mesa, CA 92626

Software Taxation

Dear *DDJ*,

I read with interest the edi-

Feel the power of VAX/VMS® for a lot less...

PCVMS.™

If you've written applications software on a VAX computer system, you already know about the power and flexibility of DEC's VMS operating system. Now, PCVMS puts the features of VAX/VMS on the IBM PC.

Designed specifically for applications programmers, PCVMS is a true multitasking, multiuser operating system that supports the most popular programming languages with resources you won't find in any other system for the same price. As powerful as it is easy to use, PCVMS offers a rich set of commands with easy-to-remember names like ALLOCATE, ASSIGN, COPY and DELETE. Like VAX/VMS it has a HELP command, which provides on-line documentation about commands, services and other topics. It also has built-in command prompting, so if you don't give all the information required for a command, the system will prompt you for the information you left out.

In addition to its powerful command set, PCVMS has over 80 system services, including enhanced system services that can be called from high level languages like BASIC, Fortran, Pascal, and C, as well as from assembly language. Using these enhanced services, you can create multiuser database, spreadsheet, and word processing programs that take full advantage of VAX/VMS system services. At the same time, PCVMS gives you multilevel protection that prevents services called by one process from inadvertently affecting or destroying other processes.

What it finally comes down to is this: if a VAX computer is out of your reach, now you can reach out and grab the power of a VAX for your PC with PCVMS.

PCVMS. From Wendin. Only $99.

Ask about our other products for the IBM PC and true compatibles.

PCNX™
True multitasking, multiuser operating system similiar to AT&T's popular UNIX® operating system.

OPERATING SYSTEM TOOLBOX™
Complete software construction set that lets you build your own multitasking, multiuser operating systems.

XTC®
The ultimate programmer's editor. Multitasking macro language plus multiple linkable windows and buffers.

All products priced at $99 with source code included.

DEALER INQUIRIES WELCOME

Foreign orders inquire about shipping. Domestic orders add $5.00/1st item, $1.00 each additional item for shipping, handling, and insurance. We accept Visa/MC, American Express, COD, and Bank Drafts drawn on U.S. Banks.
Washington residents add 7.8% sales tax.

WENDIN®
BOX 266
CHENEY, WA 99004

ORDER HOTLINE
(509) 235-8088
(MON.-FRI., 8-5 PACIFIC TIME)

VISA MasterCard

The people who make quality software tools affordable.

Circle **no. 112** on reader service card.

(continued from page 10)

torial and the lead letter in the July 1986 *DDJ*. The former complains that software is not enough of a "thing" that its sale should be subject to sales tax. The latter observes that software is enough of a thing that we can talk of the rights of ownership in it.

A software author's primary concern should be that the state recognize and protect his relationship to the thing he has created. To the extent that a state protects this relationship, the author has a "property" interest. If the author transfers all or part of that right to another person, he has transferred a thing, albeit an intangible one. Once the state recognizes the existence of a thing and defines the rights that a person has in that thing, assorted laws can come into play. The state may invoke its criminal laws to protect the owner from wrongful deprivation of the right to use and enjoy the software. The owner may invoke conversion, trespass, or a host of other civil actions to recover exclusive use and enjoyment or to force the wrongdoer to pay for his deed. The alternative is to hold that property rights do not attach to software. If the state will not protect any person's relationship to a given piece of software, it may be freely "transferred" in voluntary and involuntary (unilateral) transactions.

Once we determine that property has been transferred in a commercial transaction, the decision to apply or not to apply a state sales tax becomes a policy decision. Perhaps there are reasons that the states should not tax commercial transactions in software. I find the blatant assertion that it is "grossly unfair" to be unconvincing. It is not immediately evident that a sales tax on software is more or less invidious that the same tax on file cabinets or electricity.

My own crusade is to persuade legislatures and courts that laws should be written and interpreted to break away from the eighteenth-century notion that the law protects only tangible things. For that reason I applaud laws and regulations that treat software as a thing in which individuals can have property rights. Those of us who create and use software stand to gain much more than we lose.

Simon B. Buckner
1605-D Jefferson Heights
Jefferson City, MO 65101

The Last Square Root Letter?

Dear *DDJ*,

The July 1986 Letters column contained a letter by Dorothy Wolfe that concerns my "Square Roots on the NS32000" article published in The Right to Assemble, March 1986. Wolfe has a criticism to which I would like to respond.

Wolfe points out that, according to my definition of the (integer) square root as the smallest of two nearly equal factors, "the square root of 17 (or of any prime number) would be 1." The following should apply to any prime number; I will use the example of 17.

The problem seems to involve the meaning of the word *integer*. In the context of integer arithmetic, the term *integer* implies a number that may have a nonzero decimal component, but the decimal component is not represented—that is, 17 divided by 4 equals 4. In the context of prime numbers, the term *integer* implies a number that has a zero decimal component that is represented. In this context Wolfe's comment is valid. The only integer factors of 17 are 1 and 17—that is, 17 divided by 4 equals 4.25.

However, there are many factors of 17—1 and 17, 2 and 8.5, 3 and 5.666 . . . , 4 and 4.25, 5 and 3.4, 6 and 2.833 . . . , for example. The two "most nearly equal factors" above are 4 and 4.25, of which 4 is the smallest and is therefore the integer square root of 17. The "exact" square root of 17 is 4.12310 . . .—this is, of course, another factor.

Richard A. Campbell
198 Washington Hwy.
Snyder, NY 14226

Update

Dear *DDJ*,

Let me compliment you on your commitment to providing in-depth coverage of MS-DOS C compilers. I eagerly awaited this year's August offering and read it with great interest and in great detail.

I discovered that "Benchmarking C Compilers" by Richard Relph et al. unfortunately provided uneven coverage of the important issues and omitted information about many of the advanced features of Mark Williams Co.'s C compiler products.

A few of the product features omitted from the discussion of MWC's C Programming System were:

1. Full support for recent extensions to C, including *void*, *enum*, and structure rule extensions.
2. A powerful make utility and a Unix-style *cc* command that provides one-step compiling and linking and accepts wildcards.
3. An automated install procedure that provides unparalleled ease of installation.
4. An environment variable that provides full search path capabilities.
5. An assembler, linker, and archiver, all included at no extra charge.
6. A set of advanced, Unix-style file utilities, including diff, egrep, cmp, sort, tail, pr, and many others.

Mark Williams Co. provides cross compilers, C compilers, and operating systems to many of the largest computer manufacturers for 8086, 68000, Z8000, Z80, VAX, and other types of hardware.

Barry D. Bowen
Mark Williams Co.
1430 W. Wrightwood
Chicago, IL 60614

We will publish updates about the C compilers we have reviewed as often as is necessary. The DDJ Electronic Edition on CompuServe will contain even more information about these products.—eds.

Correction

Dear *DDJ*,

Ray Duncan's July 1986 16-Bit Software Toolbox contains a statement that Computer Innovations' Version 1.31 C compiler is now in the public domain. This is incorrect. As I'm sure you realize, the ramifications of announcing free software to the public, when in truth it is not, only adds to the problem of preventing software piracy. It also is quite distracting for us because readers have been calling and asking for copies.

I appreciate your cooperation in this matter.

Keith Wimberley
Computer Innovations Inc.
980 Shrewsbury Ave.
Tinton Falls, NJ 07724
(201) 542-5920

DDJ

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...

SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...

Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...

Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...

We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...

C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...

At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...

Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

- ☐ the C compiler for MVS software developers
- ☐ the C compiler for CMS software developers
- ☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

Name ______________________

Title ______________________

Company ______________________

Address ______________________

City ______________ State ________ ZIP ________

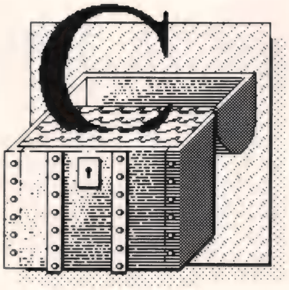
Telephone ______________________

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000

DDJ 11/86

C CHEST

Sets and Microsoft C, Version 4



Sets in C

I've been writing a book on compiler design of late. Unlike most such books, this one is going to include a lot of code. It will explain how various algorithms work by presenting real code to implement those algorithms. Not wanting to secrete all this useful stuff until the book is finally published, I'm going to publish an occasional excerpt from time to time, starting this month.

Many of the operations involved in compiler writing, such as creating state-machine tables from regular expressions, involve operations on sets, and C, unlike Pascal, doesn't have a built-in set capability. Fortunately, it's not too hard to implement sets in C by means of bit maps—as several people pointed out to me when bit-map routines were first published in this column (*DDJ*, June 1985). In fact, the Pascal implementations I've seen actually use bit maps to implement their sets. The bit-map routines that were printed last year aren't quite general-purpose enough for real set applications, so I've expanded them into the routines presented this month.

To use the set routines, you have to *#include* set.h (Listing One, page 58) at the head of your program. Most of the set functions are macros that evaluate to workhorse-function calls. The macros and subroutines are shown in Tables 1 and 2, page 20.

All the elements in the set must have numeric values, though in many instances any arbitrary number will do. Enumerated types are almost ide-

by Allen Holub

al for this purpose, though *#defines* can be used too. For example:

```
typedef enum
{
    JAN, FEB, MAR,
    APR, MAY, JUN,
    JUL, AUG, SEP,
    OCT, NOV, DEC
}
MONTHS;
```

creates 12 potential elements of a set. You can now create two sets called *winter* and *spring* by using the set operations:

```
#include <set.h>

SET *winter, *spring;

winter = newset( );
spring = newset( );

add( JAN, winter );
add( FEB, winter );
add( MAR, winter );
add( APR, spring );
add( MAY, spring );
add( JUN, spring );
```

Set operations can now be performed using the other macros in set.h. For example, *disjoint(winter,spring)* evaluates to true because the sets have no elements in common; *equivalent(winter,spring)* evaluates to false for the same reason. A third set that contains the union of *spring* and *winter* can be created with:

```
half_year = newset( );
union( half_year, winter, spring );
```

Intersection(half_year, winter, spring); creates a null set because there are no common elements. The *test( )* and *main( )* routines in set.c (lines 261–332 of Listing Two, page 62) contain additional examples.

Complemented sets present a particular problem. You'll notice that the eventual size of the set doesn't have to be known when the set is created. The set size is just expanded as elements are added to it. This can cause problems when you complement a set because the complemented set should contain all possible elements except those that are in the equivalent, uncomplemented set. For example, if you're working with a "language" that's composed of the set of symbols {A, B, C, D, E, F, G} and you create a second set {A,C,E,G} from elements of the language, the complement of this second set should be {B,D,F}. That is, the complement should be all the symbols in the language except those that are in the original, uncomplemented, set.

All sets are represented as bit maps, and these maps are of finite size. Moreover, the actual size of the map grows as elements are added to the set. You can complement a set by inverting the sense of all the bits in the bit map, but now you can't expand the set's size dynamically any more (at least not without a lot of work). To guarantee that a complemented set contains all the potential elements, you have to first expand the set size by adding an element that's one larger than any possible legitimate element and then complement the expanded set. A second problem has to do with extra elements. The bit-map size will usually be a little larger than the number of potential elements in the set. If you just complement bits, you will effectively add members to the set. On the plus side, set operations (*union, intersection*, and so on) are much easier if you physically complement the bits in a map.

An alternate method of complementing the set is to have negative-true sets and positive-true sets. Here you can just mark a set as negative or positive by setting a bit in a header. You don't have to modify the bit map at all. When you test for member-

Unleash The Most Powerful Development Tools On The Planet DOS.

UNIFY DBMS/DOS. The UNIX World Leader Brings A New Dimension To DOS Application Development.

What happens as the DOS world expands? As a new generation of hardware takes over? As networking becomes more important? The potential is enormous. But until now, the tools to achieve it have been limited.

Now a leader from another world unleashes that potential: UNIFY® DBMS. The leading relational DBMS in the UNIX™ world. And now, the most advanced set of application development tools in the DOS world.

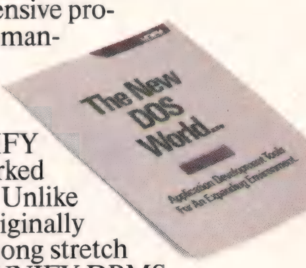
With UNIFY DBMS, DOS developers have new power to build more sophisticated applications than ever before possible.

The power to write high performance "C" programs that will access the data base, using Unify's Direct Host Language Interface.

The power of an industry standard query language—SQL.

The power of unmatched speed in production applications. Only UNIFY DBMS is specifically engineered for transaction throughput. With unique performance features like PathFinder™ Architecture multiple access methods, for the fastest possible data base access.

The power of comprehensive program development and screen management tools. Plus a state-of-the-art fourth generation report-writer.



What's more, with UNIFY DBMS, the potential of networked applications becomes a reality. Unlike DBMS systems which were originally single-user (and which have a long stretch to accommodate more users), UNIFY DBMS is a *proven* multi-user system.

And because UNIFY DBMS/DOS is the best of two worlds, it offers you the most powerful benefit of all: DBMS applications that can grow as your needs grow. From single user DOS. To networked DOS. To multi-user UNIX. All without changing your applications.

Call the Unify Information Hotline for our free booklet: The New DOS World. (503) 635-7777

See us at
COMDEX/Fall '86
November 10-14, 1986
Las Vegas Convention Center
Las Vegas, Nevada

UNIFY CORPORATION
4000 Kruse Way Place
Lake Oswego, OR 97034

Circle no. **176** on reader service card.

Tech Personal Computer's

Tech PC Twin Multiuser
Starting From $1699

Tech PC/XT base unit with 640K, and two 360K disk drives.

Two high resolution monitors, two selectric style Hi-Tek keyboards, 50 feet of shielded cable to separate the two stations.

System supports up to six printers.

Full software support with multi-level file, security, electronic message facility to send and receive messages between users, password logon system, and system operator command level.

System supports all popular software such as Wordstar, dBaseIII, Lotus 123, Multimate, etc.

Tech PC Triad Multiuser
Starting From $2599

Tech PC/XT base unit with 640K, and two 360K disk drives.

Separate Intel 80188 microprocessor running at 8 MHz and 512K for each terminal.

Three high resolution monitors, three Selectric style Hi-Tek keyboards, 50 feet of shielded cable to separate the three stations.

System expandable to 32 workstations.

System supports up to six printers.

Full support for multitasking multiterminal use with print spooling for multiple printers, background monitoring of the system, dial up bulletin board support, passwork protection, and file/record locking supporting PC Network protocol.

System supports all popular software such as Wordstar, dBaseIII, Lotus 123, Multimate, etc.

THIRD PARTY MAINTENANCE
AVAILABLE THROUGH
MOMENTUM SERVICE CORPORATION

IBM, IBM XT are registered trademarks of International Business Corporation.

Multiuser Systems. . .

Tech PC Quad Multiuser
Starting From $4499

Tech Turbo PC/AT base unit in portable or desktop configuration with 512K, multiple serial ports, three Tech PC terminals, connecting cables, and networking software.

Four users expandable to nine users over dumb terminals or PC's with terminal emulation software.

Capacity for up to 16 printers at remote sites with up to 6 local printers attached to the main unit.

Each user can access 512K of RAM with memory expansion boards.

Full support for multitasking multiterminal use with print spooling for central or terminal printing, background monitoring of the system, dial up bulletin board support, password protection, and File/Record locking using PC etwork protocol.

System supports all popular software such as Wordstar, dBaseIII, Lotus 123, Multimate, etc.

Tech PC Turbo Quad Multiuser
Starting From $5999

Tech PC/XT base unit in portable or desktop configuration with 640K, multiple serial ports, three Tech PC terminals, connecting cables, and networking software.

Separate NEC V20 8088 Intel compatible 8 MHz CPU and up to 1 MB RAM for each terminal on the system.

Two fully functional serial ports per terminal.

Four users expandable to 32 users over dumb terminals or PC's with terminal emulation software. Capacity for unlimited number of local printers.

Full support for multitasking multiterminal use with print spooling for multiple printers, background monitoring of the system, dial up bulletin board support, password protection, and File/Record locking supporting PC Network protocol.

System supports all popular software such as Wordstar, dBaseIII, Lotus 123, Multimate, etc.

TECH PC 714/385-1711

1911 Betmor Lane, Anaheim, California 92805

Telex 272006 Answer Back - TECH FAX: 714-5568325

Users only circle **no. 279** on reader services card. Dealers only circle **no. 245** on reader service card.

C CHEST

(continued from page 14)

ship, if a set is marked negative-true, you can just reverse the sense of the test (evaluate to true if the requested bit is not set). Though this takes care of all the size problems, operations on negative-true sets are much harder to perform.

Because the two representations are both useful but in different applications, I decided to implement both methods. The *invert(d,s1)* macro performs a one's-complement on all bits currently in the set's bit map. Note that if new elements are added, the new bits won't be complemented. You should always expand a set out to the maximum number of elements (by adding and then removing the largest element) before inverting it. The *complement(d)* macro implements the second method discussed earlier. It doesn't modify the bit map at all; rather it sets a bit in a header to mark a set as negative-true.

Because there are two different classes of sets (those that are complemented and those that are inverted), there are also two different macros for testing membership. *Ismember(x,s)* evaluates to 1 only if the bit corresponding to the requested element is actually set to 1. *Ismember(x,s)* can't be used reliably on complemented sets. The *test(x,s)* macro can be used with complemented sets. If a set is complemented, the sense of the individual bits is reversed as part of the testing process. If the set isn't complemented, *test( )* works just like *ismember( )* does (though it's a little larger and takes a little longer to evaluate).

Note that the various set operations (*union*, *intersection*, and so on) are valid only on inverted sets. The *set_op( )* routine ignores the complement bit in the *SET* header, treating all operands as if they were positive-true sets. Use *invert( )* if you're going to perform subsequent operations on the inverted set.

Implementation

Sets are represented as *SET* structures, defined on lines 9–17 of Listing One as:

```
typedef struct
{
    unsigned            nbytes : 13 ;
    unsigned            compl : 1 ;
    int                 nbits;
    unsigned char       *map;
    unsigned char       defmap[DEFBYTES];
}
SET ;
```

The set itself is represented as a bit map, pointed to by the *map* field of the structure. Initially *map* points at the *defmap* array. If more elements are added to the set than can fit into *defmap*, a larger bit map is allocated automatically and *map* is made to point at the larger bit map. This way you don't have to worry in advance about the maximum size of a set. The bit-map size is automatically increased as the set grows larger. The bit map is not made smaller if elements are removed, however. *Nbytes* and *nbits* keep track of the number of bits and bytes in the map. *Nbits* is always *nbytes * 8*. The *compl* field is used to mark a set as negative-true (it's 1 for negative-true sets, 0 for positive-true sets).

The *add*, *remove*, *ismember*, and *test* macros all access the bit map di-

ACTOR WILL DO WINDOWS.

If you've been wondering how you're going to explore Microsoft® Windows and then work it into your programs, what you need has just arrived.

ACTOR™

ACTOR is a new programming system, and there are two reasons why it's so good with Windows.

First, ACTOR is an ideal programming environment. It shows you all your work through Windows. Any part of the program you're writing, its output as it runs, error diagnostics and tracing, catalogs of routines, and, in fact, just about anything you like – they can all be visible on the screen and accessible at the same time, in their own windows.

So it's easier to see what you're doing.

Second, ACTOR is an object-oriented programming language. Everything in an ACTOR program is a self-contained package with its own data and procedures. That is, an object. And the objects pass messages to each other while the program runs.

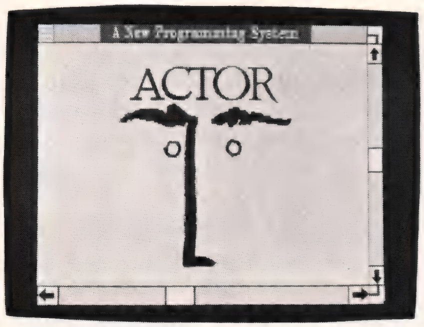
With ACTOR, you don't write lists of instructions. Instead, you define what classes of objects do. Which is especially easy because they inherit most of their behavior from other classes. That means there's less for you to worry about.

And when you create an object, you just choose its name and starting values. After that it pretty much takes care of itself.

What's important here is that windows are objects like everything else in the language. You make as many as you need, and they do what they're supposed to, mostly on their own.

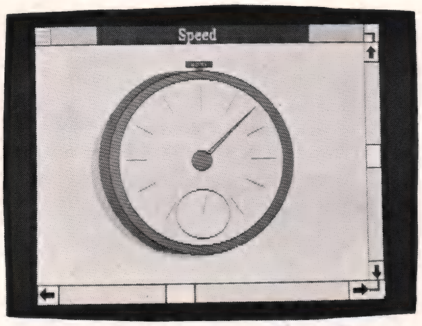
So it's natural and easy to put windows in your application programs. And users will benefit from Windows when your programs run, as much as you do when you write them.

To order, just send in the coupon. Or call **312-491-2370** for more information. You'll see that ACTOR will do Windows, and more.

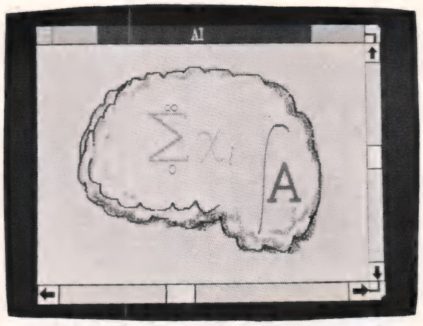


ACTOR™ is a new, interactive programming system, the first with Microsoft® Windows. You have Windows when you write a program, and users have Windows when they run it.

Moreover, ACTOR is a new, object-oriented programming language. Winding objects up and turning them loose is a lot easier, more productive, and more fun than old fashioned programming.



Thanks to a new method of "incremental garbage collection," ACTOR never has to slow down to clean up memory like other advanced languages. You can even use it for real-time control. It's that fast.



ACTOR offers all the features of an ideal artificial intelligence language, but in a familiar, Pascal-style syntax. Which makes artificial intelligence programming easier. And what could be smarter than that?

Please send me more information about ACTOR.

Name____________________

Address____________________

City__________ State______ Zip______

Daytime phone__________

☐ Check or money order enclosed. ☐ Bill my VISA. ☐ Bill my MasterCard.

Card number______________ Exp. date______

☐ I'd like to place an order while I'm at it. (ACTOR is not copy protected and has a 60-day money back guarantee. Educational pricing is available. ACTOR requires an IBM PC/XT or AT or equivalent. 640K and hard disk is recommended.)

☐ ___ copies of ACTOR with Windows runtime support at $495 each. $______

☐ ___ copies of ACTOR, including Windows complete with MS-DOS Executive, at $550 each. $______

■ Please add $5 for shipping within the U.S., $30 overseas. $______

■ Total. $______

MICROSOFT WINDOWS
AN APPLICATION DESIGNED FOR MICROSOFT WINDOWS

The Whitewater Group®
Technology Innovation Center
906 University Place, Evanston, IL 60201

Circle **no. 282** on reader service card.

C CHEST
(continued from page 18)

rectly. They are all passed a bit number and a pointer to a *SET*. Most of the work is done in the *GBIT* macro (on line 46), which is passed a set pointer, bit number, and an operator. The *((s)−>map)[(x) >> 3]* part of the macro selects the proper byte of the bit map. The right shift is a divide-by-eight. Note that the right shift would have caused problems if I hadn't defined the bit map as *unsigned char*. There would have been sign extension if the high bit happened to be set. The second part of the macro, *1 << ((x) & 0x07)*, creates a mask that corresponds to the requested bit by shifting the number 1 *x % 8* bits to the left by (the *MOD* is done with a bitwise AND operation here). Note that it's the number 1 that's being shifted, not the contents of the bit map. The actual operation is then performed by applying a specific operator to the mask. If the operator is |=, a bit is set in the map. If the operator is &, a bit is tested for true. If the two operators &= and ~ are passed, then a bit is cleared.

The three macros that use *GBIT* test to see if a bit is legal (if the bit number is too large, 0 is returned). If the number is in range, they invoke the *GBIT* macro, passing it the correct operator. *Add* is an exception. If the requested bit number is too large, it evaluates to a call to the subroutine *addset( )*, which increases the size of the set and then invokes the *GBIT* macro to set the proper bit in the newly expanded bit map (*addset* is on lines 84−94 of Listing Two). Note that *test* just evaluates to an *ismember* invocation, inverting or not inverting the result depending on whether *(s)−>compl* is true. Also note that had I said:

```
(s)->compl ? ! ismember(x,s)
                 : ismember(x,s)
```

the macro would expand to almost twice as much code as it does in its current form.

The assignment operations all evaluate to *set_op( )* calls, and the test operations all evaluate to *set_cmp( )* calls. *Set_op( )* is on lines 181−229 of Listing Two. It goes through the bit map, one byte at a time, performing various bitwise operations as needed. A bitwise AND does an intersection operation, OR does a union, and so on. Note that symmetric difference is an exclusive-OR (*x* is an element of *set1* and is not an element of *set2*). *Set_op( )* normalizes the set sizes before the operations are performed. That is, all three sets will be made as large as the largest of the three. This normalization can cause problems if you use the *invert* operation because *invert* just reverses the sense of all bits in the map. This means that elements are effectively added to the set if its size has been increased and these elements will all have a zero value.

The *set_cmp( )* routine (Listing Two, lines 121−152) is a little trickier than *set_op( )*. It also normalizes the set sizes and goes through the bit maps one byte at a time. If the *while* loop on line 140 terminates with *disj* still set to 0, then the sets are exactly equivalent (the test on line 142 will fail for all bytes in the map). The exclusive-OR operation on line 144 is being used as a bitwise not-equals operator. The test evaluates to 0 only if no two bits in the same position in both bytes are set.

void SET	delset (s) *s;	Deletes a set created with a previous *newset( )* command.
SET	*newset ( )	Creates a new set and returns either a pointer to the set or *NULL* if there wasn't enough memory.
int SET	num_ele (s) *s;	Returns the number of elements in the set—0 if the set is empty.
int SET	set_cmp (s1, s2) *s1, *s2;	The workhorse function used by the *equivalent( )* and *disjoint( )* macros. Compares two sets; returns 0 if they're equivalent, 1 if they're disjoint, 2 if they intersect but aren't equivalent.
void int SET	set_op(op, d, s1, s2) op; *d, *s1, *s2;	Another workhorse function, used by the various other macros defined in Table 2. You should use these macros rather than calling this function directly.
int SET	subset (s1, s2); *s1, *s2;	Returns true if *s1* is a subset of *s2*. Always returns true when *s1* is empty.

***Table 1:** Set subroutines*

assign(d,s1)	Copies *s1* into *d*.
clear(d)	Clears all elements of *d* yielding the empty set.
complement(d)	Complements set *d* (see text).
difference(d,s1,s2)	*d = s1 ^ s2* (symmetric difference).
disjoint(s1,s2)	Evaluates to 1 if *s1* and *s2* are disjoint (have no elements in common).
equivalent(s1,s2)	Returns 1, if the two sets are equivalent.
invert(d,s1)	Does a one's complement of all bits in bit map.
fill(d)	Sets all elements of *d* to 1.
intersection(d,s1,s2)	*d* = the intersection of *s1* and *s2*.
union(d,s1,s2)	*d* = the union of *s1* and *s2*.

The following four macros have side effects. Don't use ++, −−, or subroutine or macro invocations and so on as either argument.

add(x,s)	Adds a member, *x*, to set *s*.
remove(x,s)	Removes element *x* from set *s*.
ismember(x,s)	Evaluates to true *if x* is a member of set *s*.
test(x,s)	Like *ismember( )* but works on both complemented and non-complemented sets (see text).

***Table 2:** Set macros*

The Microsoft C Compiler, Version 4.0

First the good news. I've finally received my copy of Microsoft C, Version 4.0, and am quite pleased with it. It's a significantly better product than is Version 3.0; in fact, all the problems I had with Version 3.0 have been addressed in 4.0. All the bugs that I know about (including ones that various readers mentioned to

First High-Speed DBMS written exclusively for C applications is also Transportable

"db_VISTA™ lets you easily build complex databases with many interconnected record types..."
Dave Schmitt, President, Lattice, Inc.

High-Speed data retrieval and access, are the result of db_VISTA's unique database model. db_VISTA is designed exclusively for C application programmers and developers. Now your applications can retain key benefits of C such as speed, transportability and efficiency.

Independent Benchmark proves High-Speed model 2.76 times faster

An independent software developer benchmarked db_VISTA against a leading competitor. Eleven key retrieval tests were executed with sequentially and randomly created key files.

*Result of 11 Key Retrieval Tests

db_VISTA	:671.24 seconds
Leading Competitor	:1,856.43 seconds

db_VISTA's high-speed database model lets you precisely define relationships to minimize redundant data and only those functions necessary for operation are incorporated into the run time program.

Transportable DBMS Applications with db_VISTA

To give your applications maximum transportability every line of code for db_VISTA is written in C and complete source code is available. db_VISTA operates on most popular computers with over 10 operating systems supported. So whether you write applications for micros, minis, or mainframes...db_VISTA is for you.

How db_VISTA works...

db_VISTA uses your C compiler to build the run-time program. The programmer's interface to db_VISTA is syntactically consistent with the C language.

To use db_VISTA design then compile the schema with the database definition language processor (DDLP). Test your design with the Interactive Database Access program provided. Design your C application with appropriate calls to db_VISTA. Compile and link your C program with the db_VISTA run-time library, and your application is ready to run.

Multi-user and LAN capability

Information often needs to be shared. db_VISTA has multi-user capability and supports simultaneous users in either multi-tasking or local area networking environments, allowing the same C applications to run under UNIX and MS-DOS.

db_QUERY™ lets you ask more of your database

db_QUERY is a linkable, SQL-based ad hoc query and report writing facility. It's also royalty-free.

Royalty-Free Run-Time

Whether you're developing applications for a few customers, or for thousands, the price of db_VISTA is the same. If you are currently paying royalties for a competitor's database, consider switching to db_VISTA and say goodbye to royalties.

FREE Technical Support For 60 days

Raima's technical support program includes 60 days of free telephone support, 60 days of free software updates, and complete documentation. Technical support personnel are available to answer questions about our software or yours. Extended technical support available.

Order Schedule

	db_VISTA	db_QUERY
☐ Single-user	$195	$195
☐ Single-user with Source	$495	$495
☐ Multi-user	$495	$495
☐ Multi-user with Source	$990	$990

Not Copy Protected

30-Day Money-Back Guarantee

Try db_VISTA for 30 days and if not fully satisfied, return it for a full refund.

Call Toll-Free Today!

Order Line **1-800-327-2462**
Information Line **1-206-828-4636**

MasterCard VISA

Read what others say about db_VISTA

"If you are looking for a sophisticated C programmer's database, db_VISTA is it. In either a single or multi-user environment, db_VISTA lets you easily build complex databases with many interconnected record types. The multi-user implementation handles data efficiently with a LAN, and Raima's customer support and documentation are excellent. Source code availability and a royalty-free run-time is a big plus."
Dave Schmitt, President Lattice, Inc.

"Not 'yet another user-friendly database', it is a DBMS aimed at the technical C programmer instead of the non-technical end-user."
Hal Schoolcraft, Data Based Advisor March, 1985

"On the whole, I have found db_VISTA easy to use, very fast with a key find, and powerful enough for any DBMS use I can imagine on a microcomputer."
Michael Wilson, Computer Language September, 1985

db_VISTA Version 2.11

Database Record and File Sizes
- Maximum record length limited only by accessible RAM
- Maximum records per file is 16,777,215
- No limit on number of records or set types
- Maximum file size limited only by available disk storage
- Maximum of 255 index and data files

Keys and Sets
- Key length maximum 246 bytes
- No limit on maximum number of key fields per record—any or all fields may be keys with the option of making each key unique or duplicate
- No limit on maximum number of fields per record, sets per database, or sort fields per set
- No limit on maximum number of member record types per set

Operating System & Compiler Support
- Operating system's MS-DOS, PC-DOS, UNIX, XENIX, SCO XENIX, UNOS, ULTRIX, VMS
- C compiler's Lattice, Microsoft, DeSmet, Aztec, Computer Innovations, Xenix and Unix

Features
- Multi-user support allows flexibility to run on a local area network
- File structure is based on the B-tree indexing method and the network database model
- Run-time size, variable—will run in as little as 64K, recommended RAM size is 256K
- Transaction processing assures multi-user database consistency
- File locking support provides read and write locks on shared databases
- SQL-based db_QUERY is linkable
- File transfer utilities included for ASCII, dBASE optional

Utilities
- Database definition language processor
- Interactive database access utility
- Database consistency check utility
- Database initialization utility
- Multi-user file locks clear utility
- Key file build utility
- Data field alignment check utility
- Database dictionary print utility
- Key file dump utility
- ASCII file import and export utility

*The benchmark procedure was adapted from "Benchmarking Database Systems: A Systematic Approach" by Bitton, DeWitt and Turbyfill, December 1983.

RAIMA™ CORPORATION

3055-112th Avenue N.E.
Bellevue, WA 98004 USA
(206) 828-4636 Telex: 9103330300

Order Toll-Free
1 (800) 327-2462

High-Speed Programming Tools Transportable by Design

Circle no. 206 on reader service card.

C CHEST

(continued from page 20)

me) have been fixed. *Spawn( )* now works as it's supposed to, ^Z doesn't cause problems with the input functions or *fseek( )*, and the correct code is generated in mixed-model programs. Many of the inexplicable and hard-to-duplicate bugs in the shell have magically disappeared. I recompiled the shell and utilities with no difficulties. In fact, the shell became about 2K smaller and noticeably faster in places. Because I could dispense with the *spawn( )* work-around, batch file execution sped up by about 25 percent.

The error recovery has improved dramatically (it's now among the best I've seen), and the error messages are more informative. The *User's Guide* has been improved too. Some of the excess verbiage has been removed, and it now includes a very adequate explanation of how to use the *near* and *far* keywords.

Because the start-up module sources are now included, you can make ROMable code if you want. Microsoft says it will provide absolutely no support to aid you in this endeavor, however, so you're probably still best off using the Aztec C compiler if ROMability is an issue. The same goes if you're planning to port to another environment. None of the library source code is available from Microsoft.

The compiler comes with the usual utilities (lib, link, and so on), and a version of make is now included as well. The make is not a full implementation of the Unix make, but it's adequate (it supports generic dependencies [*.c.obj*] and the *$** and *$@* macros—the version that shipped with MASM didn't).

The most important addition to the package is the CodeView debugger. In the past, I've shied away from debuggers when I've written high-level-language programs. They're just too much work; adding a few *printf( )* statements takes less time and is more informative than messing with debuggers. Even "symbolic" debuggers

Flotsam and Jetsam

Starting this month I'm adding a new feature to C Chest, a sort of Holub's helpful hints for C programmers. Every month (hopefully) there will be a short inset article that discusses some part of the language that's liable to be useful to both beginning and advanced C programmers. If you've a helpful (and short) hint of your own, send it in. This month I'm going to look at the problem of nested comments and at how to get rid of all those unsightly *#ifdef DEBUG*s that clutter up your code.

Comments don't nest in C. Consequently, a fragment such as:

```
/*
    code( );   /* comment */
    more( );
*/
```

won't perform as expected. The */ on the second line will terminate the /* on the first line. The *more( );* subroutine will be compiled, and the */ on line 4 will generate a "missing open comment" error message. This problem is usually circumvented using the mechanism:

```
#ifdef NEVER
    code( );   /* comment */
    more( );
#endif
```

Here, provided that *NEVER* is not *#defined* anywhere, the code between the *#ifdef* and *#endif* won't be compiled. Just "never say NEVER" to quote Romeo Void. The same mechanism is used to disable debugging diagnostics, as in:

```
#ifdef DEBUG
printf("Debug diagnostic");
#endif
```

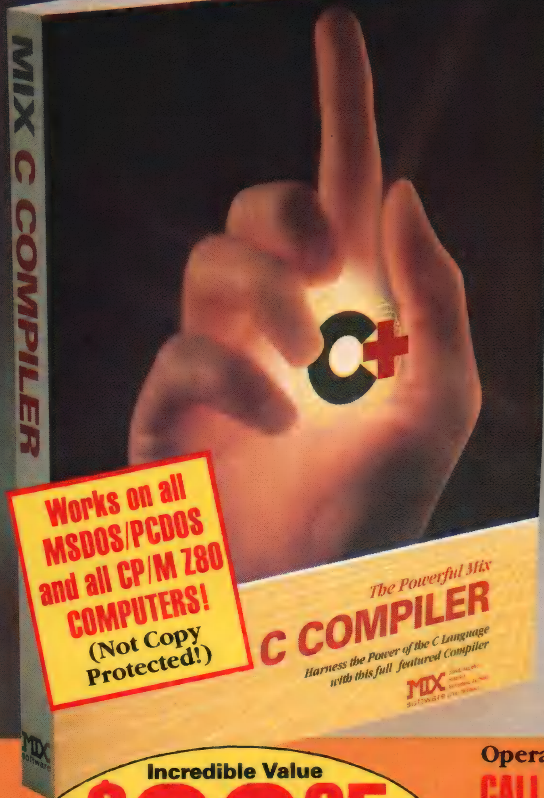
I always seem to delete a diagnostic message five minutes before I need to use it again. An *#ifdef* lets me get rid of it without actually deleting it.

A problem here is code readability. For portability reasons the # must be in the leftmost column and there can be no space between the # and the *ifdef*. Consequently, the *#ifdef*s mess up all your careful indenting. Moreover, three lines are now required for every debugging diagnostic. These problems can be solved by using the macro mechanism more intelligently. Consider the following:

```
#ifdef DEBUG
#define D(x) x
#else
#define D(x)
#endif
```

If *DEBUG* is *#defined*, then the *D( )* macro expands to its own argument. If *DEBUG* is not *#defined*, then the *D( )* macro expands to a null string, to nothing. That is, the entire *D(arg)* macro invocation, along with the argument, is ignored. The argument can be any legitimate C operation. For example, *D(printf("hi");)* expands to *printf("hi");* when *DEBUG* is *#defined*. The whole statement, including the *printf( )* call, is discarded when *DEBUG* isn't *#defined*. Because *D( )* is a macro expansion, as compared to a definition, none of the pound-sign-must-be-in-the-left-column restrictions apply. The *D* can be at any indent level. An example of the *D( )* macro is in this month's Listing Two on line 65.

There are three caveats. First, don't put a semicolon after the *)* in the macro invocation. A semicolon by itself is a legitimate statement in C (it doesn't do anything but it's legal). If there's a semicolon following the *)*, it will still be around even if the macro expands to a null string, and it can cause problems with *if/else* statements binding incorrectly. Second, be careful of the comma operator. This is not much of a problem because the comma operator isn't used very often, but the C preprocessor can't distinguish between a comma operator and the comma that separates macro arguments. You'll probably get an error message if you use a comma operator inside a *D( )* invocation. Finally, many compilers won't accept macro invocations that are longer than one line, so the entire macro should be on a single line.



C FEVER *catch it!*

It's becoming an epidemic . . . everyone is switching to C! First there were a few hackers, then came the college students, next the major software houses, and now the rest of the programming world. Programmers everywhere are infected with the desire for SPEED, POWER, and PORTABILITY.

It's time to face the inevitable. You're going to catch the fever too! When you do, give us a call. We've got the best cure—an illustrated guide to the C language, plus a complete program development system. Everything you need to master the C programming language . . . all at a price that's less than the cost of a book!

But don't let this price fool you. Our system is powerful; it compiles twice as fast as the others, is completely standard, and it's very easy to use. Most C compilers are designed for wizards. We have designed ours for you!

What do you get for a mere $39.95?

- A 450 Page book filled with sample programs, plus . . .
- A fast, standard, full featured C compiler that supports all data types and the latest features like bit fields, enumerations, structure assignment, and passing/returning structures.
- A fast linker that loads separately compiled files, searches libraries, and builds an executable program.
- An extensive library of more than 170 functions (including the standard C functions and the computer specific functions that provide direct access to the operating system and BIOS).
- Tools that allow you to optimize your programs for minimal space or maximum speed.

Incredible Value AT ONLY $39.95 With 30 Day Money-Back Guarantee

Operators are standing by . . . Please use this Number for ORDERS ONLY!

CALL TOLL FREE FOR RUSH ORDER DELIVERY!

1-800-523-9520

IN TEXAS, PLEASE CALL TOLL FREE 1-800-622-4070

For Technical Support Please call 1-214-783-6001

MIX Software, Inc. / 2116 E. Arapaho / Suite 363 / Richardson, Texas 75081

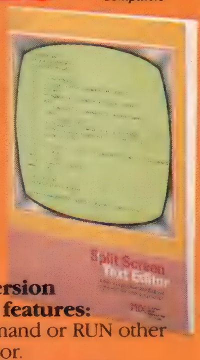
Or contact our Worldwide Distributors direct in:

Canada:	Saraguay Software	1-416-923-1500
Australia:	Techflow	047-586924
Switzerland:	DMB Communication	CH-1-825-53-29
France:	Info/Tech	1-43-44-06-48

Split Screen Text Editor

an Incredible Value AT ONLY $29.95

(Not Copy Protected!) Works on all MSDOS/PCDOS and CP/M Z80 Computers



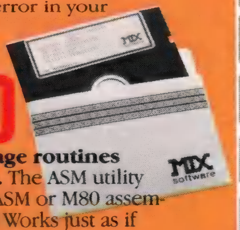
Our high powered editor is great for editing high level languages. **It works just like Micropro's Wordstar,™** but macros allow you to create your own custom editor, and the split-screen feature lets you edit two files at the same time.

The MSDOS/PCDOS version is loaded with special features:

- Execute any DOS command or RUN other programs from the editor.
- Quickly edit files as large as 300,000 characters.
- Compile MIX C programs directly from memory. The editor automatically positions the cursor to the first error in your program.

ASM UTILITY

an Incredible Value AT ONLY $10



Call assembly language routines from your C programs. The ASM utility works with Microsoft's MASM or M80 assembler. Macros make it easy! Works just as if you were calling a C function, and you can even call C functions from assembly language. Lots of useful assembly language functions are included as examples.

RUSH REPLY ORDER FORM!

Limited Time Only $54.95 C Compiler & Text Editor

Description	Quantity	PRICE	Total Order
Split-Screen Text Editor	______	$29.95	$______
C Compiler	______	$39.95	$______
C and Text Editor (Special)	______	$54.95	$______
ASM Utility	______	$10.00	$______
Texas Residents Add 6.125% Sales TAX			$
Shipping Charges (See at Right)			$
TOTAL OF YOUR ORDER:			$

Please check method of payment:
☐ Check ☐ Money Order ☐ MasterCard/VISA

Your Card #: ______________________

Expires ______________________

Shipping Charges: (No charge for ASM Utility)
In the U.S.A.: Add $5.00 per Order.
In CANADA: Add $10.00 per Order.
OVERSEAS: Add $10.00 per Text Editor. Add $20.00 per C Compiler. Add $30.00 for combined C Compiler and Text Editor.

Operating System: ***(Check one)***
☐ CP/M Z80 ☐ MSDOS/PCDOS

Specify Your Computer Name ______________

Specify Disk Format ______________

NAME ______________

Telephone A/C (______) ______________

Street ______________

City ______________

State ______________

Country ______________ ZIP ______________

MIX software
2116 East Arapaho
Suite 363
Richardson, Texas, 75081

Ask about our Volume Discounts!
Call 1-214-783-6001

D

Circle no. 156 on reader service card

C CHEST

(continued from page 22)

aren't that useful because they don't give you access to local variables.

CodeView is a whole other ball game. It's wonderful. First of all, it's a true source-code debugger. The source code is right there in front of you on the screen. You can set breakpoints in the source code, page around in the code with either a mouse or the cursor keys, even go look at other files if you like. If control passes to a subroutine in another file, the second file is read automatically so the correct source is still displayed. You can do things such as position the cursor at a line in the source code and then execute up to that line with the push of a button. If you want to see the assembly language, push F3 and it's right there in front of you (with the source code interspersed as comments, no less). Push F2, and a registers window appears (which remains active while the code is running so you can actually see the registers change). The line being executed is highlighted in reverse video so the control flow is visible in front of your very eyes. In slow execution mode, you can actually watch a *for* statement loop and you can watch control skip over an *else* clause.

Among the nicest features of CodeView are the "watch" functions. A watch window can be opened in which contents of variables (even local variables) are displayed. You can watch the values change as the program executes. You can set a "watch point," a breakpoint that stops execution when an expression evaluates to false (the expression uses the normal C operators and can include any of the local or global variables). You can also set a "tracepoint," a breakpoint that stops execution when a specific memory location or range of memory locations is modified (unfortunately there's a 128-byte limit on ranges so you can't say "break if any of my code space is modified"; you can't have everything). You can even do all this simultaneously—variables changing, registers flipping, control flowing all at once—it's like a Christmas tree.

There's also a variable-evaluation feature that's quite nice, again letting you use C syntax. For example, if *s* is a structure that contains a pointer (*p*) to another structure that contains an array (*c*), you can ask CodeView to show you *(s.p)->c[3]* and it does it. You can say *p* to see the contents of a pointer, and you can say **p* to see the object pointed to. You can also use *printf*-like format specifiers. For example, a number can be displayed in hex by putting *,x* after the name. The contents of a string pointer can be displayed as a number, or you can actually see the string (by putting a *,s* after the name). You can also modify variables easily using the C operators: *a++* and *a=10* work as expected.

The subroutine-calling stack is available in symbolic form. You can see the entire subroutine calling sequence, with both the calling subroutine's names and the values of all the subroutine's arguments as part of the display. It even works with recursive subroutines.

This is a great debugger.

Now for the bad news. The compiler's price has gone up to $450, an amount at which the anarchist programmer in me rebels. You do get a lot for your money, but that's still a

C CHEST
(continued from page 25)

lot of money. More important, the cost of an update ($150) is, I think, unreasonably high considering the number of bugs in Version 3.0. On the other hand, CodeView may well be worth the $150 all by itself.

I found three bugs in the package: a serious one in CodeView and a couple of trivial ones in the compiler itself. When I first starting using the debugger, it had an annoying tendency to go off into outer space occasionally. The mouse cursor disappeared, and the keyboard wouldn't respond to anything. I had to press Ctrl-Alt-Del to get out. I suspect, because this error stopped happening once I learned how to use the debugger, that it's incorrect handling of a command syntax error that's at fault.

Another problem seems to be with the mouse itself. I'm using a Microsoft mouse (the older version of the hardware but with the new driver). My friend Bill, who uses the Logitech mouse, reports that CodeView crashes when he tries to use his mouse. Though Microsoft assures me that, if a mouse driver is Microsoft compatible, it should work, the compiler package does come with a new version of the driver and this driver may have new or undocumented features. There's no documentation provided for the mouse driver. As the Logitech mouse works fine with Microsoft Windows, I can't help but think that it's CodeView that's at fault, not the driver. Though a mouse isn't required to use the debugger, this is one application in which the mouse is actually pretty nice. It's a real indication of CodeView's strengths that both Bill and I continue to like it in spite of these problems.

I found two bugs in the compiler itself, both trivial. The code:

```
#define isquote(c) ( (c)=='"' ||
                     (c)=='\'')
```

incorrectly generates the warning "warning 74: non standard extension used - 'macro formals in strings.'" This message is somewhat obtuse, and it's not explained at all in the error-messages appendix. Because the appendix stops with warning 72, I assume there's also a warning 73 that isn't explained either. I think the compiler is complaining about a macro such as:

```
#define printnum(n,t) printf("%t", n);
```

In this example, the call *printnum(10,x)* should expand to *printf("%x", 10)*. This situation doesn't apply to the actual code, however; the compiler is probably confused by the double quote. A work-around is:

```
#define isquote(c) ( (c)=='\"' ||
                     (c)=='\'')
```

The second bug has to do with the new, and nonstandard, *cdecl* keyword. A function that's declared *cdecl* will be compiled with C parameter-passing conventions even if the compiler command-line switch that forces Pascal or FORTRAN parameter-passing conventions is used. It should be a no-op if this command-line switch (*/Gc*) isn't specified. If *LINT_ARGS* is *#define*d at the head of your file, the following definition is included in signal.h:

```
int cdecl (*signal(int, int (*)( )))( );
```

If, later on in the same file, you include the line:

```
extern int ( *signal(int, int (*)( )))( );
```

then the error "error 86: 'signal' : redefinition" is generated. It seems as if the *cdecl* keyword is not really a no-op if */Gc* isn't given. As I often explicitly declare all external subroutines at the head of a file, this is forcing me either to use a nonstandard keyword or always to compile with the */Za* command-line switch. This second alternative also disables the *near* and *far* keywords, however. I'd prefer the *cdecl* keyword to actually be ignored if */Gc* isn't used on the command line.

The situation is made worse by an omission in the *User's Guide* index. There's only one entry for *cdecl*, pointing at page 193. *Cdecl* is mentioned on this page, but there's no description there of what it does. A reference to page 204, where the keyword is actually explained, should be added to the index.

In summary, I like the compiler and I really like CodeView. If Microsoft would only publish a regular newsletter telling us about known bugs in the compiler as they're discovered, provide better support (for start-up module modifications for example), and provide the library sources if you need them . . . .

Erratum: A Bug in Sort

There's a bug in the sort program printed in this column in June 1986. The program isn't closing input files when it's finished with them, so the number of simultaneously open merge files is unnecessarily limited. To fix the problem, insert an *fclose(fp);* statement just above the *if* statement on line 385 of Listing One (sort.c). You'll have to put curly braces around the body of the *while* because it now has two statements in it.

A reader, David Schuler, also caught an error in the sort article. The example on page 22:

```
 2
 1
 20
 10
```

won't sort as explained because of the leading white space. The example should be written:

```
2
1
20
10
```

Availability

The code from this month's column is available on CompuServe, and an IBM PC-compatible disk is available for $25 from Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705. The newly compiled version of the shell (2.01) is available from *DDJ* (see advertisement, page 108). This version also corrects the *ESC* environment and shell-variable modifier bugs found in Version 2.00. Updates from earlier versions are available from *DDJ* for a $6 media charge.

DDJ

(Listings begin on page 58.)

Vote for your favorite feature/article.
Circle Reader Service **No. 1**.

APDA was founded by A.P.P.L.E. CO-OP, one of the first and largest Apple users' groups. We've got the full cooperation of Apple Computer, Inc.

For just $20 per year, we can give you all of the above. Plus a data-packed quarterly catalog/newsletter, update bulletins and access to our on-line ordering system.

If you join APDA before 1987, you'll receive your choice of a free Apple II or Macintosh technical book worth more than the price of admission.

So call now.

206-251-6548.

Or write: Apple Programmer's and Developer's Association, 290 SW 43rd St., Renton, WA 98055.

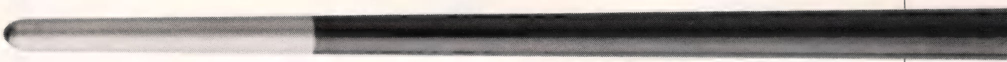
We can't guarantee you'll write the same great code as *some* of our members.

But it won't be because of lack of information.

Apple Programmer's & Developer's Association

Circle **no. 231** on reader service card.

New Issues in PC Graphics



by Ed McNierney

Display boards incorporating third-generation graphics controllers have provided software developers with a highly sophisticated set of capabilities. Along with those capabilities comes a level of complexity previously unknown in the PC graphics industry; in order to use these new hardware tools effectively, programmers need to become familiar with the new issues they raise and with the new techniques associated with those issues.

The new graphics chips are true microprocessors that run in parallel with the host CPU.

The chief features of these new controllers, which include the Intel 82786 and the Texas Instruments 34010, include hardware graphics primitives, which give them the ability to draw circles, lines, and bit-mapped text at high speed. The controllers also have true microprocessor architectures that allow the development and execution of complex graphic algorithms in parallel with the host CPU's execution; the ability to address large amounts of memory (up to 512 megabytes) in support of multiple pages of screen memory and large amounts of off-screen graphics storage memory; and hardware windowing capabilities that, for the first time, completely separate the physical layout of graphics memory from the data displayed on the screen.

Two Examples

Although they are both graphics coprocessors, the Intel 82786 and the TI 34010 are not head-to-head competitors in the graphics market. The 82786 is able to address 4 megabytes of memory and display that memory in hardware windows at a resolution of up to 640×480 pixels.

Ed McNierney, Number Nine Computer Corp., 725 Concord Ave., Cambridge, MA 02138

Internally, the 82786 is structured as three distinct processors: a graphics processor that performs drawing, a display processor that extracts bit-map data from memory and generates a display from it, and an interface unit that mediates requests for access to display memory. The graphics processor is capable of executing a simple command list stored in graphics memory. Although the command set supports subroutine call and jump instructions, there are no conditional branch opcodes. The command list, therefore, cannot be executed intelligently and must be constructed and managed by an external intelligent processor—usually the host CPU. These features combine to make the Intel 82786 an excellent choice for general-purpose, high-resolution graphics systems, such as might be used in business graphics, graphic user interfaces, and desktop publishing systems.

The architects of the TI 34010 have taken a slightly different approach. The 82786 is capable of only simple (but very fast) drawing and complex display manipulation, whereas the 34010 is the reverse. It provides no hardware windowing, and it is only capable of generating a display that shows different portions of graphics memory by splitting the screen into horizontal strips. It does not have as high a clock speed as does the 82786, but it is capable of executing extremely complex graphic algorithms. Its strength lies in drawing rather than in display generation.

Algorithms for drawing complex figures can be coded and executed directly by the 34010, which operates as a 32-bit processor capable of addressing 512 megabytes of memory. It is therefore suited to high-resolution, drawing-intensive applications such as computer-aided design, drafting, and high-end publishing and page composition.

Optimized for Graphics

The execution of graphics primitives by either of these graphics processors provides two direct benefits: Unlike the host CPU, the graphics processor is designed to draw efficiently and quickly, and the presence of a coprocessor frees up the host to perform other tasks while the drawing is being performed. Current host CPU instruction sets are optimized toward the manipulation of numeric and string objects, not graphics; they allow instructions that update pointers to move through linear blocks of memory (such as strings or buffers) rather than through rectangular areas suitable for graphic applications. CPUs are being forced into a service they were never designed to perform, whereas graphic processors address memory properly and also contain silicon implementations of drawing algorithms. Both features greatly reduce the amount of code required to execute a figure. Listing One, page 66, for example, contains three versions of a routine that draws a 10×10 rectangle—one on a CGA in an 8086 system, one in an 82786 system, and the last in a 34010 system. Note that in the second two cases, the host processor is not executing the algorithm but is instead free to prepare for the next graphic output sequence. A system incorporating a graphics processor can have its software tuned to achieve real-time performance increases of 50 to 100 times compared to a host CPU/display buffer system.

For cases in which the graphics primitives are insufficient for the task at hand, the TI 34010 provides a true microprocessor instruction set, with arithmetic instructions, conditional test and branch instructions, and software and hardware interrupt control. Not only does this provide the developer with a general-purpose coprocessor but it also permits the 34010 to be used as the only processor in the system. Intelligent terminals or printers can be driven by only one CPU, reducing hardware cost and simplifying software development. Although the TI processor is slower in terms of clock speed and in the types of memory accesses it can perform than the Intel 82786, its programmability can allow a greater synergy between it and the host CPU, resulting in greater application throughput. A graphic processor is not a panacea that will cure a sluggish graphic implementation, however, because the task of creating a system in which both processors are used optimally is an extraordinarily difficult one. The complexity of such coprocessor systems is suggested in Figure 1, page 32, in which sample 34010 and sample 82786 system memory layouts are compared.

Both the TI and Intel processors are capable of addressing large amounts of graphics memory. Although more memory is required simply to support high-resolution and high-color displays, these processors require access to additional objects in their memory—graphic software (both in ROM and in RAM), memory-mapped register sets, control areas, and graphics source data all reside in the processor's address space. One of the more difficult issues facing developers is the management of all this memory because neither the Intel nor the TI processors supports explicit memory management other than reserving certain portions of their address spaces for some specific uses.

Graphics memory also has to accommodate many new types of objects whose presence springs directly from the fact that they support off-screen graphics data. No application developers really want to store fonts, icons, cursors, and temporarily hidden portions of the display in system memory; they have, however, been forced to do so because there has been no other place to put them. All the housekeeping efforts that currently have to be exerted in the host address space in order to keep track of these objects must now be transferred to the graphics processor. The host is equipped with the operating system's normal memory allocation, deallocation, and management functions, all of which need to be duplicated and executed by the graphics coprocessor to manage its own memory. Although display memory grows in complexity, it is at least performing a task for which it is intended, freeing system memory for the execution of host CPU code and the storage of host CPU data.

The management of graphics memory becomes espe-

(continued from page 31)

cially complex in the case of the 82786 because it has full hardware windowing support. As graphics-based user interfaces become more common, the 82786 will prove a great benefit to application developers and to the underlying windowing systems that support them. An 82786-based system is entirely freed from any association between physical memory addresses and screen displays—any graphics data residing anywhere in its address space can be displayed at any point on the screen by creating a window, positioning it on the screen, and defining the bit-map address from which that window should take its data. This system provides greater flexibility for the application as well as increased performance in windowing environments. On an IBM Enhanced Graphics Adapter, for example, moving a 400×300-pixel rectangle in 16-color mode to another position on the screen requires moving 60,000 bytes of data—and then moving up to 60,000 bytes more to fill the hole uncovered by the moved rectangle. In an 82786 environment, however, the same operation requires updating only about a dozen bytes, performing the same task up to 10,000 times faster!

The fundamental benefit derived from a hardware-based windowing system is that the covering of one window by another does not destroy any display information but simply hides it. Moving the covering window then reveals that graphic data again, without the application having to store a temporary copy of it and replace it—panning, smooth scrolling, and the repositioning of windows all become much faster and much more practical to implement in graphical user interfaces. Not only do these windows save graphics memory by obviating the need for maintaining duplicate copies of display data but they also save memory by allowing data in different windows to be displayed at different pixel depths.

In a high-color paint package using an 8-bit-per-pixel display, there is no need for a pop-up window containing a list of picture file names to be displayed in anything but black and white. By creating a new 1-bit-per-pixel window in which to display the text, the application reduces both the memory required for the display and the time required to print the text in the window by a factor of 8. This greater flexibility means that display memory resources can be used more prudently and in a manner more appropriate to the particular display application at hand.

New Expertise Is Needed

All these wonderful advantages are not without their price. As with any new hardware or software technology, a whole new area of expertise has to be developed. The examples set by the Apple Macintosh and the IBM

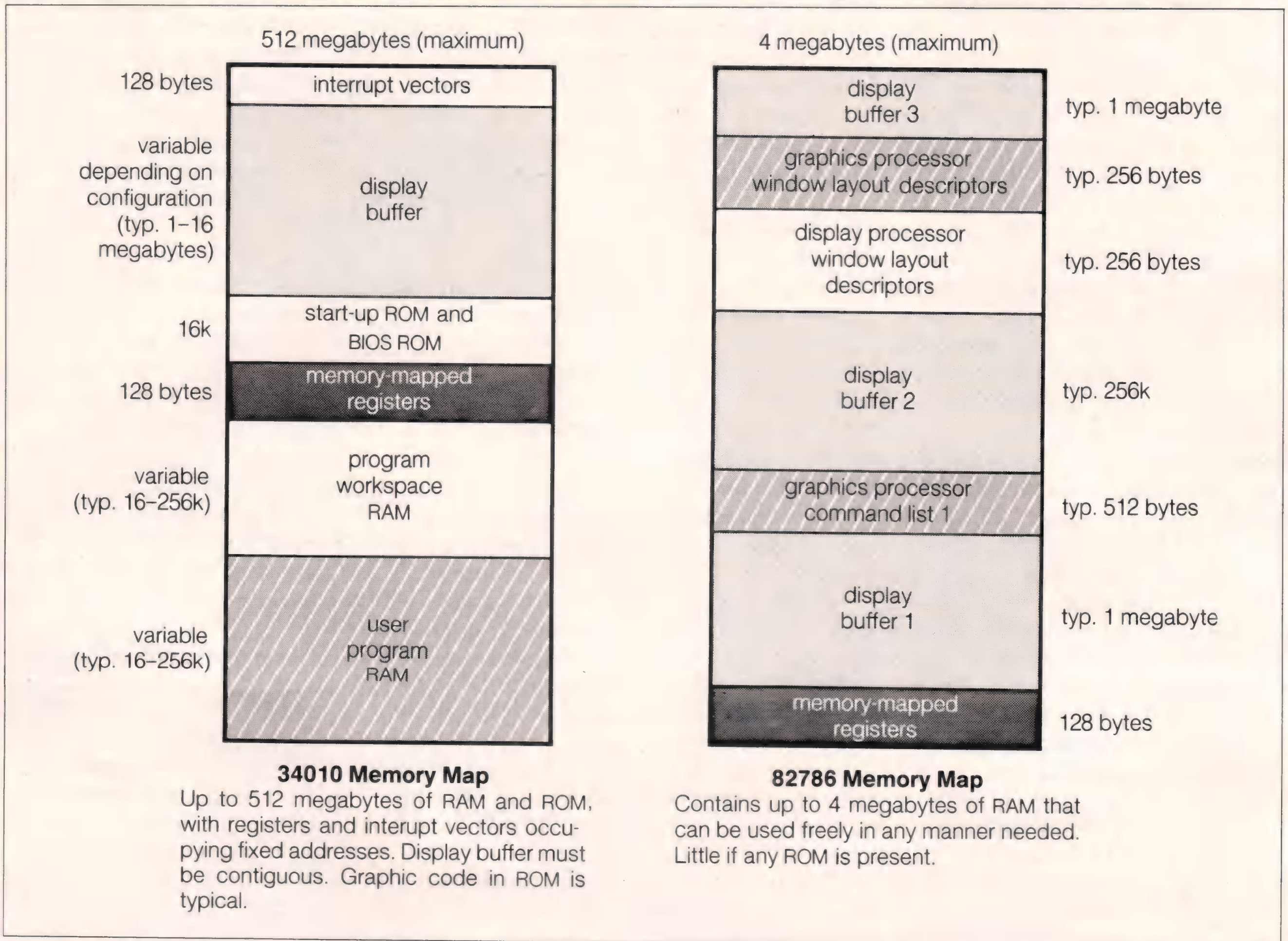
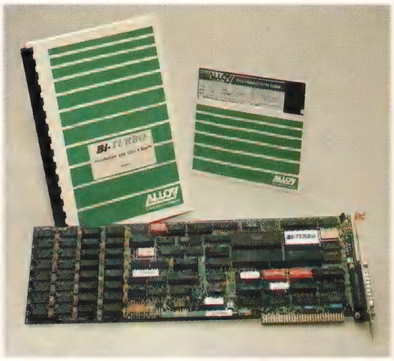


Figure 1: *Comparison of sample 34010 and 82786 system memory layouts*

When You Need To Do Two Things At Once — Bi-TURBO™ . . . from ALLOY



Alloy's Bi-TURBO combines the features of today's best-selling boards and adds distinctive Alloy engineering experience to create the first accelerated dual-tasking card for personal computers. Bi-TURBO is the *complete* dual-tasking solution, with everything you need to be twice as productive.

- Adds power to PCs, XTs and even ATs
- Adds a full 640 KB of "second task" RAM, with its own 8 MHz processor
- Adds 256 KB of RAM-based disk cache to speed disk access for both tasks
- Easy-to-use dual tasking software — switch *full screen* windows with a keystroke!
- Print spooler, with *multi-printer* capability

With One PC and Bi-TURBO, Do the Work of Two PCs

You're in the middle of a three-hour mail merge and you need to run a spreadsheet — NOW! Bi-TURBO to the rescue! Press one key on your keyboard and you're running the spreadsheet — while the mail merge task continues to run *at full speed.*

You get both jobs done, and Bi-TURBO's RAM-based disk caching actually SPEEDS UP the mail merge!

$995! — Bi-TURBO Saves Money and Time.

Bi-TURBO gives you the features and benefits of multi-function cards and windowing software — *and much more performance* — for less.

Bi-TURBO is the *complete* power upgrade for your XT or AT, with added 8 MHz processing power, added RAM, disk caching, print spooling, truly *easy-to-use* dual-tasking software and optional 8 MHz 8087 math co-processor.

Call your local distributor today . . . and Bi-TURBO.

Crystal Computers, Inc.
Lenaxa, KS 66214
(913) 541-1711
Irving, TX 75063
(214) 929-1300

FA Components
Elmhurst, NY 11373
(718) 507-1444
Ft. Wayne, IN 46808
(219) 432-8540
Greenville, SC 29607
(803) 288-2422

First Source Distributing
Salt Lake City, UT 84119
(801) 973-0074

Micro Computer Distributors, Inc.
Huntington Beach, CA 92649
(714) 895-5841

Micro Wholesalers, Inc.
Hunt Valley, MD 21030
(301) 666-5300

PC Distributing, Inc.
Northbrook, IL 60002
(312) 298-1400

PGI Corporation
Tempe, AZ 85281
(602) 967-1421

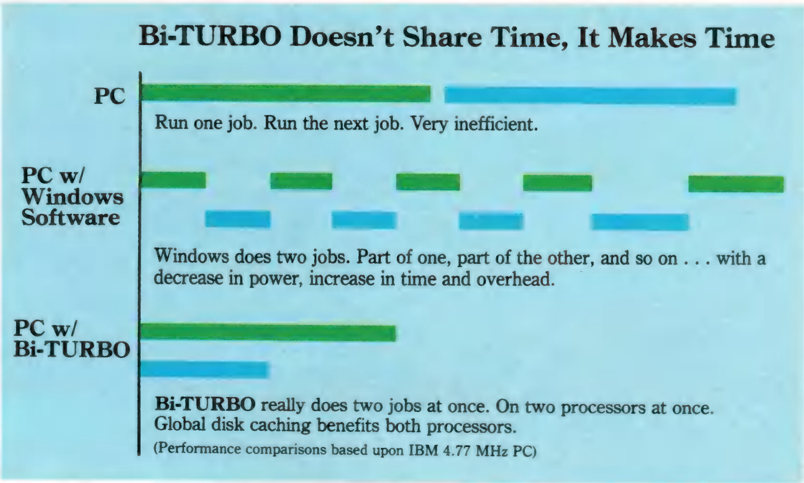
Super Source
Norcross, GA 30071
(404) 441-3451

Vitek
San Marcos, CA 92069
(619) 744-8305
San Jose, CA 95131
(408) 436-8026

Vitronix Corporation
Westboro, MA 01581
(617) 366-1144

W4 Micro Distributors
Birmingham, AL 35209
(205) 945-8310

Western Micro Systems
Scottsdale, AZ 85260
(602) 948-4240
Mountain View, CA 94043
(415) 964-2050
Northridge, CA 91324
(818) 700-9922
Redmond, WA 98052
(206) 881-6737



Windows is a Registered Trademark of Microsoft Corp.

ALLOY Computer Products, Inc.

Alloy Computer Products, Inc., 100 Pennsylvania Avenue, Framingham, Massachusetts 01701. (617) 875-6100, TWX: 710-346-0394
Alloy Computer Products, Inc., 9 Executive Circle, Suite 240, Irvine, California 92714. (714) 261-7661
In Europe: Alloy Computer Products (Europe) Ltd., Cirencester, Gloucestershire, England. Tel. 0285-69571, Tlx: 43340
In Australia: Alloy Australia Pty. Ltd., Suite 3, 318-322 Stephensons Road, Mount Waverley, Victoria, Australia 3149. Tel. 011-613-277-1333

Circle **no. 178** on reader service card.

(continued from page 32)

EGA are good parallels to the graphic coprocessor environment. Both of these technologies represented significant advances over what was previously available in the market, and they both ran into dificulties because of the long lead time required for software development to take place. The new graphic coprocessor systems are complex, and it will take a while before sufficient expertise is acquired to use them well or at all.

Although software development efforts are simplified at the microscopic level in that a developer is spared one more implementation of Bresenham's circle algorithm, the overall environment is more demanding and more detailed, and the development overhead required may prevent smaller software firms from undertaking any development until a significant market leader has become apparent. The ability to take an intelligent graphics board and use it to run existing applications faster will help break the hardware/software development deadlock in which software developers won't port to new boards until they sell and hardware developers can't sell boards until the software support is there. As a result of this complexity, the first coprocessor systems will probably do nothing more than execute current software packages more quickly. There will be a definite development lag before the coprocessors are used fully and effectively for new, innovative applications.

Hindering the ability of developers to work with these products is the current state of graphic software standards. In different respects they provide too much and too little. The graphic software world has too many "primitive" standards, and a developer often has to make a difficult choice between VDIs, CGIs, GKS, and PHIGS systems, not to mention new systems being proposed. A standard just isn't a standard when a developer has to support six of them to cover 50 percent of the market! On the short side, none of the present standards addresses the issue of windowing, although an ANSI committee is working on a windowing proposal. Unfortunately, it will almost certainly take a while for standards committees to propagate new proposals for these new processors, and by that time several independent interfaces will exist and the standards will occupy the same role they do now.

Communication and Synchronization

On the more practical side, some implementation problems need to be addressed by any application that attempts to use a graphic processor well. Because each processor has its own instruction set and opcode syntax, commands need to be translated from the format in which the application program generates them to a format intelligible by the graphic processor. A well-thought-out communication syntax is necessary in order to minimize the overhead required to send the desired command to the graphic processor, or the application may end up taking longer to perform the drawing than if the host CPU had executed the graphic algorithm itself. Going hand in hand with an efficient communications protocol is a careful synchronization of the tasks being performed by the host and graphic processors. Because a PC with an 82786 or 34010 is a true multiprocessor system, the architecture of the graphics board may require full status information to be provided to the host CPU. If a command is sent to the graphics processor to draw a circle, and then the host CPU wishes to read the value of a pixel along the circumference of that circle, the host needs to be able to tell whether the circle has been completed, lest the incorrect value be read.

An extreme approach to synchronization is to wait for any command or program sent to the graphic processor to complete before returning control to the host application. Although "lockstepping" will work and will execute faster than a host CPU without the processor, the capabilities of the processor are largely ignored in this case. Synchronization schemes can include mutually interruptible systems in which each processor signals the other when it is ready for new material, critical flags that tell one processor that the other is in a possibly unstable state, and polling mechanisms in which each processor registers a request to the other and then polls for a status flag to be set indicating that the request can be honored. Figure 2, below, illustrates these schemes.

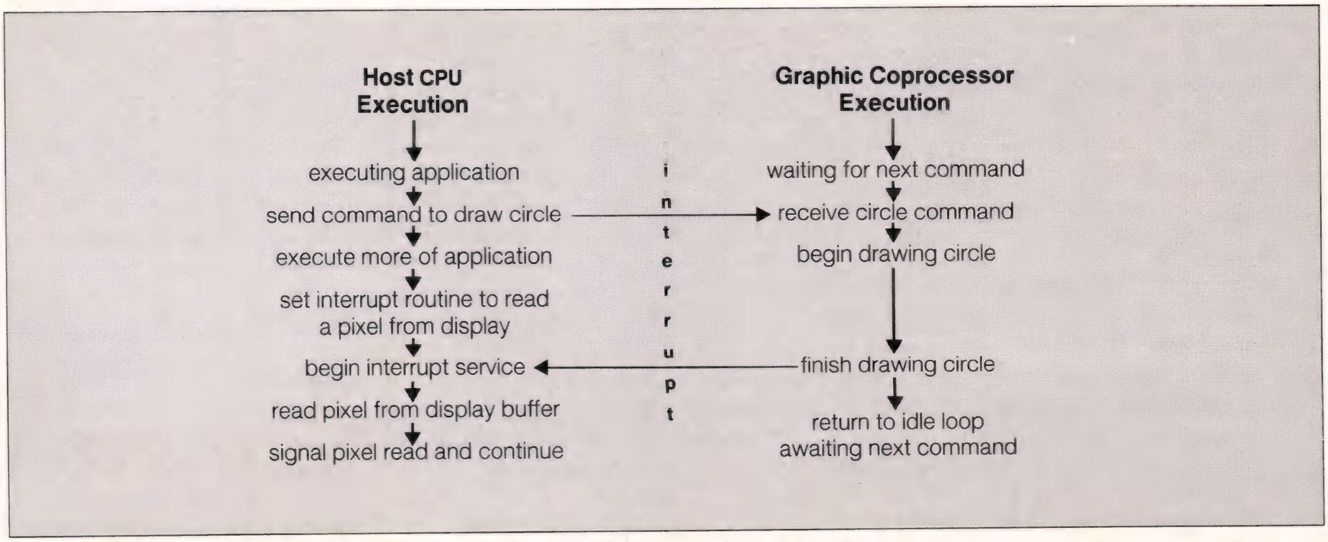


Figure 2: *Synchronization of dual processors through a mutual interrupt system*

(continued from page 35)

Black Boxes

Additional issues are brought up by the blurring of certain traditional boundaries in PC environments. There is less and less of a distinction between "graphic" and "system" memory; packages such as Microsoft Windows require that a display driver be able to perform drawing in precisely the same manner on a given block of graphic data, no matter where it resides in the system. Drawing may be performed in traditional system memory, in bank-selected expanded memory, or in on-board graphic memory.

The difficulty here comes from what in other cases is a benefit—the graphic algorithms embodied in the graphic processor are "black boxes," opaque to the user. If the graphic processor is not capable of performing drawing in system memory because of limitations of software or hardware design, then the application must provide an algorithm that duplicates the one contained in the black box in every respect. If the algorithm cannot be duplicated precisely, the graphic processor becomes useless in that system—unfortunately, the IBM PC's architecture prevents a graphics board residing in an expansion slot from modifying system memory in any way. This restriction has been lifted in the PC/AT, but it still presents an obstacle blocking access to a large installed base of machines. In addition, because the graphics board may contain far more memory than the host CPU system does, that memory may be made available for nongraphics use by other applications such as RAM disks, disk caches, and expanded memory drivers. Developers need to avoid painting themselves into corners by clinging to old assumptions that were valid on IBM Color Graphic Adapters but that are no longer appropriate.

Dealing with Text

Aiding the battle against rapidly multiplying display modes but clouding the implementation battle is the growing awareness and appreciation of the fact that text in any form is nothing more than a special type of graphic data. Text can be zoomed, scaled, colored, italicized, or displayed in many different fonts all on the screen at once, and that same text must be able to be read back from the screen. Character recognition on a monochrome screen is straightforward, but the extraction of ASCII data from a highly complex graphical bit map is a very challenging task.

The further hardware and software manufacturers can go in removing constraints on the use and appearance of text, the closer they will come to an ideal user interface. The application of multiple scaled fonts to WYSIWYG word processing is obvious, and the growing desktop publishing market will certainly benefit from these new processor technologies. Average users are becoming more sophisticated and more demanding about the text they see every time they turn on the PC. High-quality text must be provided without an excessive speed penalty on the part of users; no matter how pretty it looks, if it's not fast it won't be liked.

Both the TI and Intel processors provide sophisticated text support, allowing for color bit-mapped text to be generated at speeds approaching those of hardware character generators. Fonts can be stored in graphic memory, freeing precious system memory from the task, and special attributes such as underlining and boldfacing can be synthesized on the fly, obviating the need for storing multiple copies of the same font. Fonts are not tiny objects, and extremely large amounts of memory may be required to store and format them. The effective use of font storage may be the most critical issue in the apparent performance of a graphics application because poorly managed font memory translates into slow text performance and text performance is the one area in which users are most sensitive and critical. Judgment and clever implementation here can certainly make the difference between a popular package and a failure.

Fortunately, as the silicon available becomes more complex, the tools available become more powerful. Already function libraries that perform three-dimensional graphics, floating-point arithmetic, text generation and scaling, and object filling and shading are being provided, both by chip manufacturers and by third parties. In addition, TI even offers a C compiler for the 34010, allowing the direct conversion of algorithms developed on earlier systems. These libraries and compilers are no longer the convenient graphics toolkits offered for existing graphic display boards but are essential steps to application development.

Virtual Displays

Because it provides hardware windowing directly, the 82786 can create a virtual-display environment that directly parallels the 80386's Virtual 86 execution mode—each Virtual 86 task can think it is running on an IBM Color Graphics Adapter, while each virtual CGA is being displayed in a distinct hardware window. This chip companionship not only allows existing applications to run unmodified on an 80386 but also allows them to execute unmodified in a windowing environment (see Figure 3, page 38).

Along with the essential ability to execute current applications, the 82786 introduces the concepts of visual hierarchy and privilege in direct parallel to the software execution controls provided by the 80386. A truly integrated multitasking system must be able to treat graphic and visual information with the same protection and control as it treats any other portion of its execution environment. Imagine an 82786 graphic application running in a multitasking environment. When the application starts up, it wants to display itself in the top window, centered on the screen and occupying three-quarters of the visible area of the screen. Although it is quite reasonable to expect that the application considers itself important enough to merit all that room, the operating system knows that there are several other applications active that may require attention. By making the window positioning and sizing commands privileged system calls, the operating system can act as an intermediary between what the application wants and what resources it considers appropriate to supply. The request can be examined, modified to fit the state of the system at the moment, and the application notified of the modifications required to fulfill the request. Alternatively, the application could be

The LimeLight Computer Projector. Project Anything You Can Create on Your Computer Screen . . .

Software Demonstrations — Training — Presentations

The LimeLight is compact, complete and will operate with virtually any personal computer or terminal.

- **Projects 5-6-10 foot image**
- **Text and/or Graphics (40-80-132 columns)**
- **Simple to operate (average setup time 3 minutes)**
- **Lightweight (26 lbs.)**
- **Portable**

For more information call 1-800-331-2834 (U.S.), **1-800-521-9905** (Calif.)

limelight®
COMPUTER PROJECTOR

VIVID™
SYSTEMS INCORPORATED

41752 CHRISTY STREET
FREMONT, CA 94538

Circle **no. 247** on reader service card.

(continued from page 36)

allowed to test the manner in which a specific request would be fulfilled and modify or cancel the request entirely depending upon the results.

The concept of visual hierarchy and privilege can be the beginning of another step forward in user-interface design. By providing the means to manipulate shared bit-mapped displays in a multitasking system, a graphics processor frees application designers from one more level of hardware constraint—instead of designing to be cooperative with other applications sharing the same screen, designers can work with a true virtual system in which they appear to own not only all of memory (and then some) but also to own an entire dedicated display. This freedom, combined with the ability to produce higher-color and higher-resolution displays and the freedom gained by off-loading the host CPU from graphics work, will allow remarkable growth in the sophistication and ease of use of software; the tools are finally becoming available to catch up with the dreams of software designers.

The Next Generation

Where will the first fruits of graphic processor technology appear? Right away systems will incorporate them into high-speed versions of current products. The chief problem with graphical operating environments (Windows, GEM, and the like) is that they incur vast overhead in maintaining the screen. Ports of these environments (and similar graphics-intensive applications, such as CAD packages) to faster, graphics-processor-based systems will become available almost immediately—some may even have been introduced by the time this article is printed. Graphics toolkits and libraries, especially the popular ones such as MetaWINDOW and HALO, will follow soon after. Some of the more foresighted of these libraries have been emulating the capabilities of graphic processors in software for quite a while; applications that have taken advantage of those emulations will benefit most greatly. In parallel with the porting of graphic operating environments will come the development of enhancements to those environments, especially in the field of font and text development. The ability to generate text at high resolutions with acceptable speed will spur consumer demand for more powerful text systems and displays that begin to approach the flexibility and resolution available in their laser printers. Operating system and memory management products will, lag behind. When available, however, they will be the sources of the largest direct benefit to users.

Conclusion

Intel and TI have produced two very different but remarkable processors. The software industry will reap many benefits from them both, in part because each company has been very responsive to the concerns and needs of developers. The Number Nine Pepper Series of graphics boards includes products that use both the TI and Intel coprocessors. In addition to providing powerful hardware, the Pepper Series addresses the software issues raised here through the Number Nine Intelligent Operating System (NNIOS).

As the technology of visual displays is brought to a price and performance level that can make it available to the mass market, the opportunity for software of high perceived value increases. Public awareness of the quality and craftsmanship inherent in excellent software will certainly increase, and developers who make the necessary investments in time and effort to use these new graphic systems effectively will become a prominent part of that awareness.

DDJ

(Listing begins on page 65.)

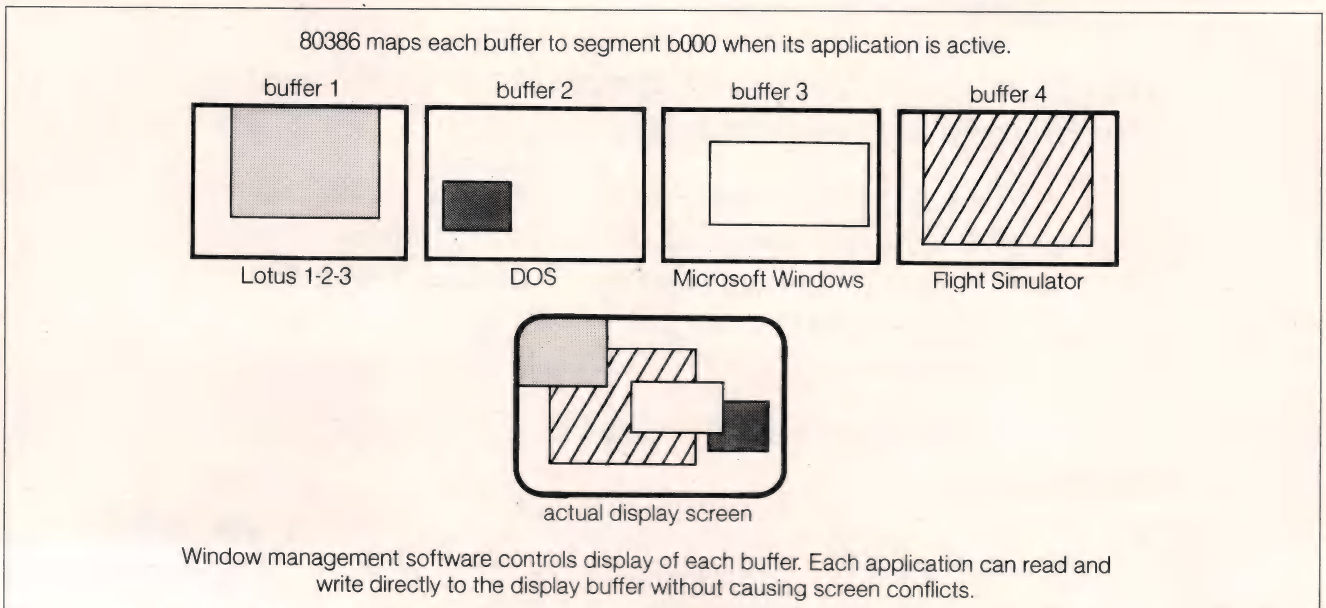
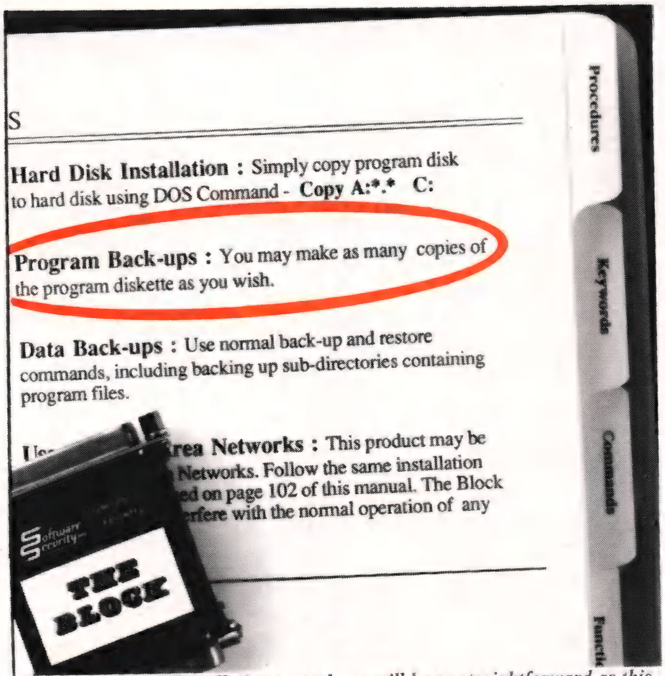


Figure 3: *82786 managing multitasking display*

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905

203 329 8870

The fastest C

Your search for execution speed is over. The new Microsoft® C Compiler Version 4.0 is here. With blazing performance. We've added common sub-expression elimination to our optimizer that produces code that rips through the benchmarks faster than ever before.

"...the Microsoft performance in the benchmarks for program execution is the best of the lot overall." —William Hunt, *PC Tech Journal*, January, 1986.*

But speed isn't the only edge you get with Microsoft C. Other advantages include a variety of memory models like our new HUGE model that breaks the 64K limit on single data items. Plus our NEAR, FAR and HUGE pointers, which provide you greater flexibility. All this allows you to fine tune your program to be as small and fast as possible.

"Excellent execution times, the fastest register sieve, and the best documentation in this review ...Microsoft Corporation has produced a tremendously useful compiler." —Christopher Skelly, *Computer Language*, February, 1986.

No more debugging hassles.
Introducing CodeView. Free.

Now, for a limited time, we'll give you an unprecedented programming tool when you buy Microsoft C, free. New Microsoft CodeView™ offers the most powerful tool yet in the war on C bugs. Forget the hex dumps. Now you can view and work with programs at any level you want. Use the program source, the disassembled object code, or

Microsoft C Compiler Version 4.00

Microsoft C Compiler
- Produces fast executables and optimized code including elimination of common sub-expressions. NEW!
- Implements register variables.
- Small, Medium and Large Memory model libraries.
- Compact and HUGE memory model libraries. NEW!
- Can mix models with NEAR, FAR and the new HUGE pointers.
- Transport source and object code between MS-DOS® and XENIX® operating systems.
- Library routines implement most of UNIX™ System V C library.
- Start-up source code to help create ROMable code. NEW!
- Full proposed ANSI C library support (except clock). NEW!
- Large number of third party support libraries available.
- Choose from three math libraries and generate in-line 8087/80287 instructions or floating point calls:
  - floating point emulator (utilizes 8087/80287 if installed).
  - 8087/80287 coprocessor support.
  - alternate math package – extra speed without an 8087/80287.
- Link your C routines with Microsoft FORTRAN (version 3.3 or higher), Microsoft Pascal (version 3.3 or higher) or Microsoft Macro Assembler.
- Microsoft Windows support and MS-DOS 3.1 networking support.
- Supports MS-DOS pathnames and input/output redirection.

Microsoft Program Maintenance Utility. NEW!
- Rebuilds your applications after your source files have changed.
- Supports macro definitions and inference rules.

Other Utilities
- Library Manager.
- Object Code Linker.
- EXE File Compression Utility.
- EXE File Header Utility.

C Benchmarks In seconds

	Microsoft C 4.0	Lattice C 3.0	Computer Innovation C 2.3	Aztec C86 3.2	Wizard C 3.0
Sieve of Eratosthenes (register)	82.9	151.4	172.3	88.0	91.9
Copy Block	86.9	231.7	199.0	123.8	189.5

Run on an IBM PC XT with 512K memory

Microsoft CodeView
Window-oriented source-level debugger. NEW!
- Watch the values of your local and global variables and expressions as you debug.
- Set conditional breakpoints on variables, expressions or memory; trace and single step.
- Watch CPU registers and flags as you execute.
- Effectively uses up to four windows.
- Debug using your original source code, the resulting disassembly or both intermingled.
- Use drop-down menus to execute CodeView commands.
- Access the on-line help to lead you through CodeView's options and settings.
- Easily debug graphics-oriented programs since program output is kept separate from debugger output.
- Keyboard or optional mouse support.
- Enter in familiar SYMDEB or DEBUG commands.

*Reprinted from *PC Tech Journal*, January 1986, copyright 1986, Ziff-Davis Publishing

you've ever seen.

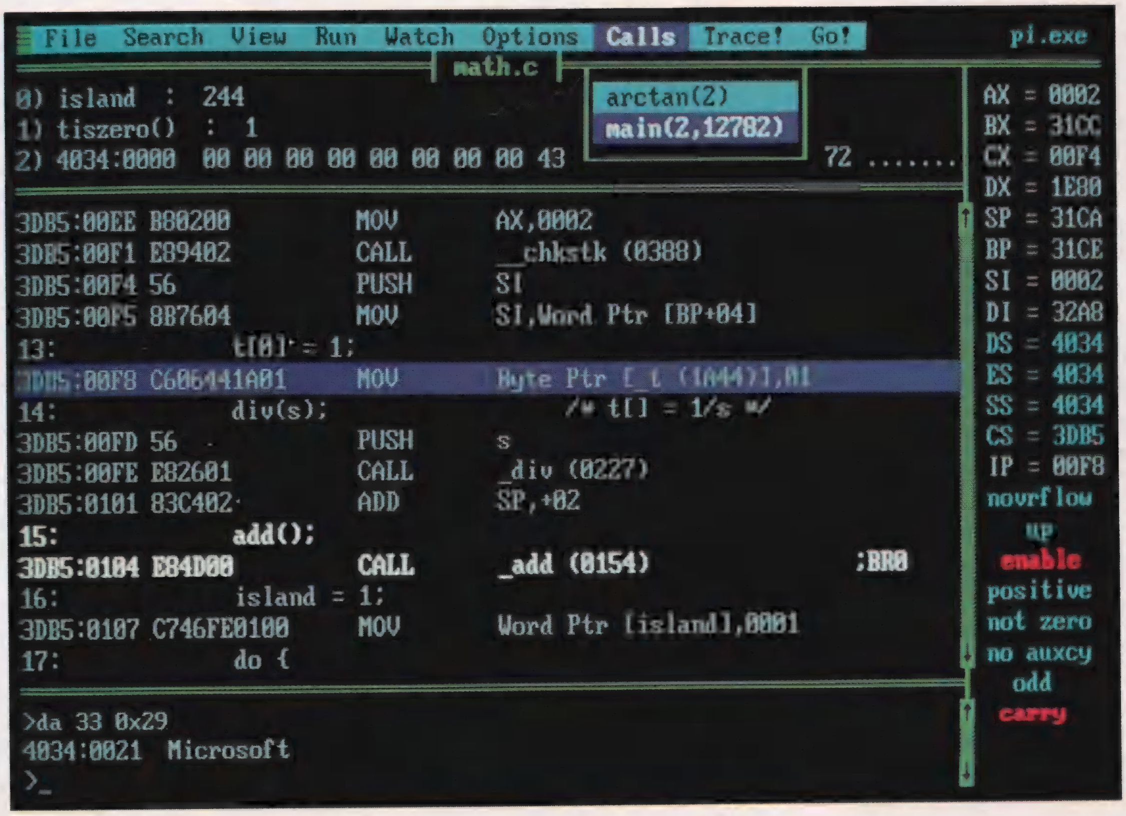
both at the same time. Open a window to view CPU registers and flags. Watch local and global variables as well. All while your program is running.

CodeView gives you complete control. Trace execution a line at a time—using source or assembly code. Or set conditional breakpoints on variables, memory or expressions. CodeView supports the familiar SYMDEB command syntax, as you'd expect. Commands are also available through drop-down menus. Combine the new window-oriented interface with our on-line help and debugging has never been easier. Or quicker.

Take the $5 CodeView tour.

You may find it hard to believe our debugger can do all we've claimed. So we're offering test drives. Five bucks will put you behind the wheel of a Microsoft C demo disk with CodeView.† See for yourself how fast debugging can get.

For more information about the CodeView demo disk, the new Microsoft C Compiler, a list of third party library support or the name of your nearest Microsoft dealer, call (800) 426-9400. In Washington State and Alaska, (206) 882-8088. In Canada call (416) 673-7638.



Microsoft® C Compiler

The High Performance Software

A Mandelbrot Program for the Macintosh

by Howard Katz

This program explores the most complex object in mathematics.

A. K. Dewdney's Computer Recreations column in *Scientific American* often provides inspiration for programmers looking for new projects. Dewdney frequently discusses interesting and offbeat algorithms and other programming matters. His column of August 1985 in particular seems to have touched off something like a feeding frenzy among hackers looking for new algorithmic adventures. In that column, Dewdney discussed the Mandelbrot set, a mathematical object named in honor of the French mathematician Benoit Mandelbrot, of fractal fame. Dewdney also provided several computer-generated images of the set, which he called "the most complex object in mathematics," that are strikingly beautiful. Interested readers might refer to Mandelbrot's classic volume, *The Fractal Geometry of Nature* (New York: W. H. Freeman & Co., 1982), for other fractal creations.

This article describes a 68000 program, written using Apple's MDS assembly-language development system, that produces screen images of the Mandelbrot set on a Macintosh. The final application is just over 4,000 bytes long. The source code, in two sections, is in Listings One and Two, pages 72 and 84. Listing One, at just over 700 lines, contains the main body of the program. Listing Two is the assembler source for a string-to-fixed-point number conversion routine that is assembled separately and then linked with the REL file produced by Listing One.

The algorithm described by Dewdney is surprisingly simple. Of the more than 1,200 lines of code in the program, fewer than 40 lines are dedicated to the actual calculations involved in the algorithm. The rest of the program is devoted to dealing with the well-known Macintosh user interface—windows, dialog boxes, and the like—and to handling the conversion and storage of the user-input parameters that dictate which area of the set to display and at what magnification.

The algorithm discussed by Dewdney involves the use of complex numbers. I'll provide a brief overview of the algorithm, but I refer interested readers to Dewdney's excellent discussion of the subject. Suffice it to say that the Mandelbrot set is the result of applying an extremely simple iterative function to each point of interest in the complex plane, where the starting value that seeds the function is the position of the number in the plane. The result of each iteration is a new complex number. If the size of the number—its distance from the (0, 0) origin of the plane—exceeds 2 at any point before the iteration runs a predetermined maximum number of times, then the point lies outside the set. If the iteration runs its full course and the size of the complex number remains less than or equal to 2, then the point lies within the Mandelbrot set. The actual iterative function involves nothing more than starting with a value of 0, adding the complex value of the point, and squaring. Each successive result is then fed back into the iterative function. Note that the terms *within* and *without* are relative: A true rendition of the set would require an infinite number of iterations; happily, you can obtain pleasing results with as few as 30 or 40 iterations per point.

Objectives

I had two major objectives in mind when I wrote this program: The first was to produce attractive and interesting images; the second was to produce them as quickly as possible. Although the algorithm is quite simple, it is also extremely computation intensive. I wanted to explore as much of the set as possible but did not want to sit around for any great length of time before being able to see the results of a session.

One final objective was to build up a library of interesting Mandelbrot vistas using the screen-dump facility of the Macintosh. In addition to storing the actual graphics image, I also wanted to be able to save all the relevant parameters so that I could reproduce the session at my leisure.

In terms of an attractive screen display, the fact that the program runs on a Macintosh immediately places it at somewhat of a disadvantage in comparison to programs written for other machines. All the MandelZoom programs (so named by Dewdney) I have seen to date use color and produce strikingly beautiful screen images. The Macintosh, of course, is a black-and-white machine. What the Mac does have is an exceedingly crisp and clean display, at a reasonable 342 × 512-pixel resolution. It also has the ability to draw using a

Howard Katz, 6989 Russell Ave., Burnaby, BC, Canada V5J 4R8

variable-size pen and with a user-selected pen pattern. Patterns take the place of colors in this implementation; I think the results shown in the accompanying screen dumps are quite pleasing. The real beauty of the Mandelbrot images lies not simply in the graphic image of the Mandelbrot set itself—the strange, beetlelike object seen in Figure 1, page 45—but in allowing the regions adjacent to the boundary of the set to be set off in different colors, or patterns if you will, depending on the number of iterations reached before the size of the complex number calculated for each point exceeds 2. Half the fun of running this program comes from varying the count "breakpoints" that determine the size of each region.

Fixed-Point Numbers

The problem of getting the program to run as quickly as possible was an interesting one. Derivation of the Mandelbrot set requires the use of real numbers because the complex values used in the computations have fractional as well as integer components. Most implementations use floating-point numbers for this purpose. On the Mac, floating-point support is normally provided by a disk-based package known as SANE, for Standard Apple Numeric Environment (now provided in ROM on the Mac Plus). SANE, however, seemed a bit slow for my purposes.

I found documentation for three routines in ROM that supported another variety of real-number representation known as fixed point. In fixed-point arithmetic, the integer portion of a number is stored as a 16-bit quantity in the high-order word of a 4-byte long word and the fractional portion is stored in the low-order word. A bit of informal benchmarking convinced me that fixed-point calculations would run roughly an order of magnitude faster than would floating-point operations, at the cost of some precision; the trade-off seemed reasonable. I didn't discard the SANE package entirely—I used its conversion routines for converting the three user-input parameters from string to SANE floating-point format, and then I converted from the single-precision floating-point format back to fixed-point representation. See Listing Two for the tedious details.

ROM Conventions

The program makes use of several of the 500-odd routines that are built into the Macintosh ROM. It's beyond the scope of this article to discuss all the details of how they are used—Caroline Rose et al.'s *Inside Macintosh* (Reading, Mass.: Addison-Wesley, 1985) documentation devotes more than 1,000 pages to that task—but a quick overview might be useful for readers unfamiliar with the Macintosh.

Most of these routines, or "traps," are dedicated to implementing the Mac user interface. Traps can be identified in the source code as identifiers preceded by an underscore, such as *_GetNextEvent* or *_PenPat*. The file MacTraps.D, which is *included* at the top of the main listing, is simply a long list of equates in which each trap name is equated to a unique, 2-byte hexadecimal value that starts with a hex $A. This makes use of the 68000 "line 1010 trap" feature (a hexadecimal $A is 1010 in binary), in which execution of any in-

MANDELBROT PROGRAM
(continued from page 43)

struction whose first nibble is a hex $A forces the processor to suspend its current operations and vector through an address in low memory to a trap dispatch table, where the following three nibbles of the instruction are decoded to determine which particular trap routine to execute. Simple, right?

Parameter passing for the ROM routines follows Pascal conventions, in which the parameters are pushed onto the stack in the order documented in *Inside Macintosh*. If a parameter is longer than 4 bytes, a pointer to its address is passed instead of the actual data. And if the routine is a function (and therefore returns a value), space must be cleared on the stack for the function result before the parameters are pushed and the result popped from the stack once the routine returns. Most of the operating system routines found in ROM do not use the above stack-passing convention; a register-based parameter-passing convention is used instead. Finally, you should note that many of the operands referenced in the program have *(A5)* suffixed to their names. This indicates that the operand in question was defined using the *DS* (define storage) assembler directive at the end of the source listing. All variables so created are referenced in code as an offset off register *A5*.

Program Description

The program uses two dialog boxes and one window. Windows and dialogs are two examples of user-interface objects that are supported by several routines in the Macintosh ROM. Although windows and dialogs can be defined in code, it is generally much simpler, and provides better documentation, for programmers to define them using Apple's RMaker (or resource maker) program. The concept of resources is, as far as I know, unique to the Macintosh, and it would take a much longer article than this to do them justice. RMaker is generally run last in the development sequence, following linking. Listing Three, page 87, is the source file that is input to RMaker for the MandelZoom program.

The first Parameters dialog allows users to select the x and y coordinates of the region to be plotted; the size of the region; and the count breakpoints, which determine what patterns are associated with what count ranges. The x and y coordinates refer to the lower left-hand corner of the drawing window, which comes up once the dialog is dismissed by clicking the Plot button. The Side parameter refers to the y coordinate of the window; the length of the image along the x axis is scaled according to the ratio of the window's width to its length. You can cycle through the input fields using the Tab key.

The first (top) count on the right side of the dialog box is the maximum number of iterations that will be performed for each point. If the program is able to iterate this number of times, the point will be drawn in a solid black pattern. If the size of the complex number produced by the iteration exceeds 2 at any point, then a lighter pattern will be used. Suitable selection of these four breakpoint count values allows users to turn one

or more of the patterns on or off or to vary the thickness of the various count "regions." Figure 1, right, for example, shows a count selection that disables all patterns except black and white for a crisp representation of the Mandelbrot set itself.

Finally, the dialog allows users to choose one of three pen sizes using the Radio buttons in the lower-left corner of the box. The default selection is for a 2 × 2 pen. In general, I use the 4 × 4 pen when exploring a new region for the first time. This provides a quick, though "chunky" plot. If the image looks suitably interesting, I continue my explorations using the finer 2 × 2 pen. The 1 × 1 pen is most suitable for producing high-quality images of the boundary of the Mandelbrot set, as in Figure 1.

Once the user clicks the OK button at the bottom of the dialog box, the Mandelbrot window appears and drawing begins. If at any time you aren't satisfied with the image being generated, you can either click on New Plot to return to the Parameters dialog or on Quit to exit from the program.

The central core of any Macintosh application is the event loop. In most Macintosh programs, the trap *_GetNextEvent* is polled continually to determine if the user has pressed a key on the keyboard or clicked the mouse (among other possible user-initiated events). In this program, the event loop is executed at the end of each Mandelbrot scan line to determine if the user has clicked either of the above buttons.

I should note one final feature of the program. In the original version of the program, the pen pattern was set and each point plotted using QuickDraw's *_Line* command as soon as the iteration for each point was completed. I found, however, that plotting ran about 20 percent quicker if I deferred the actual drawing until forced to by a change in the pen pattern. You can force the program to plot each point as it's calculated by holding down the mouse when the program first launches.

DDJ

(Softstrips begin on page 71b.)

(Listings begin on page 72.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3**.

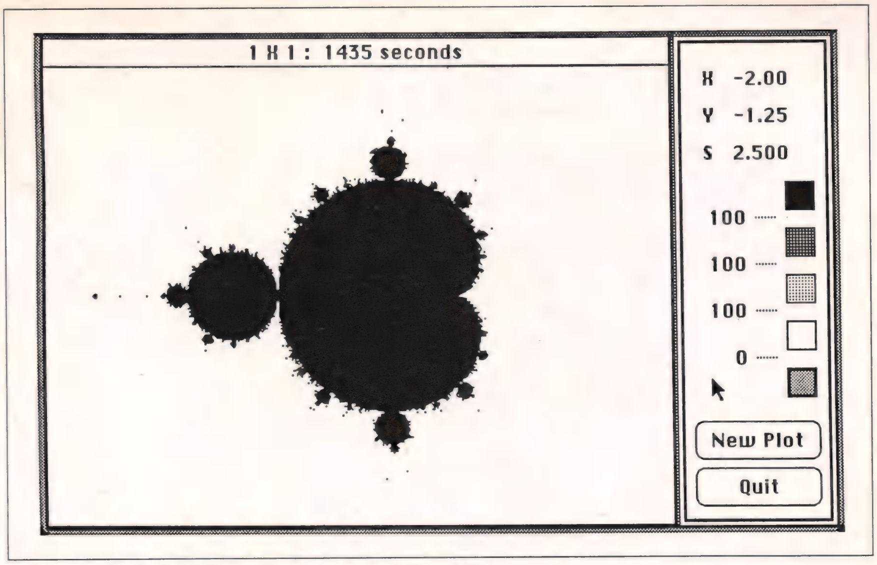


Figure 1: *Full Mandelbrot at 1 × 1 resolution*

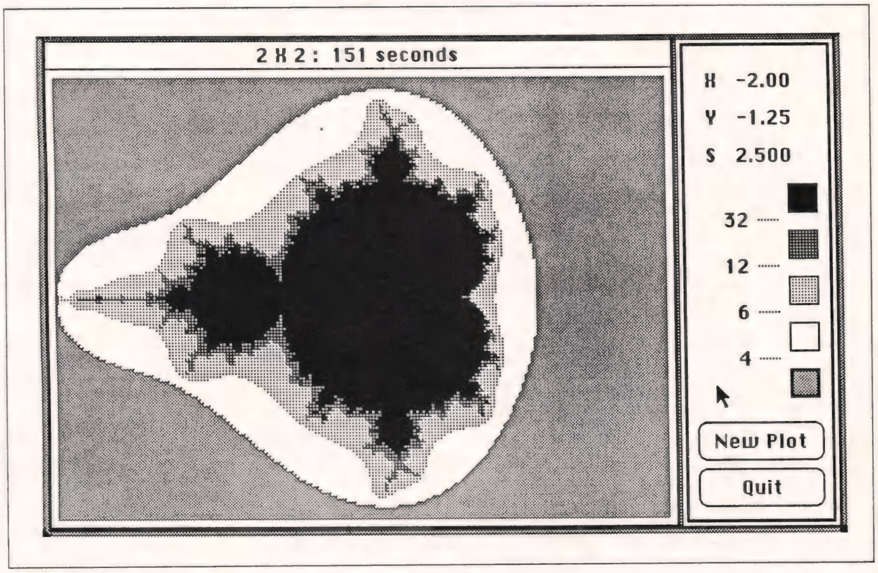


Figure 2: *Full Mandelbrot at 2 × 2 resolution*

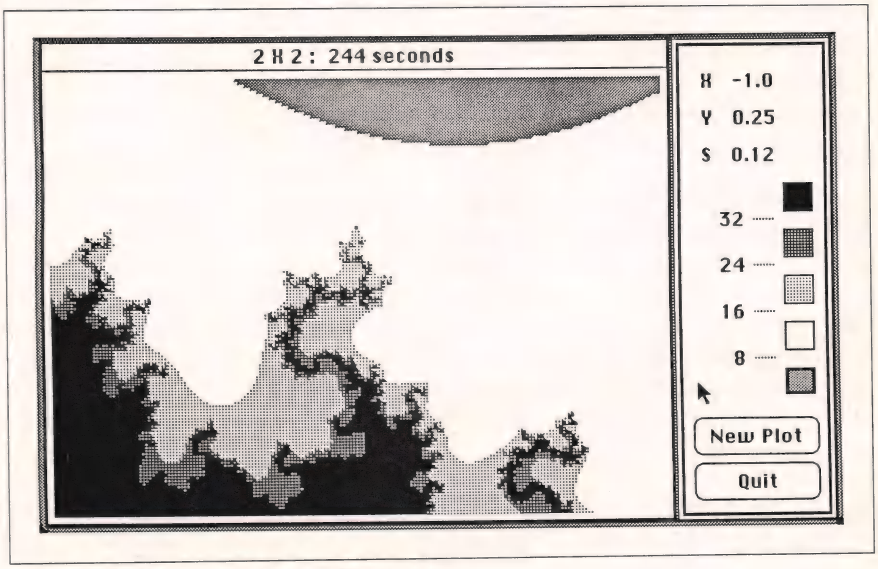
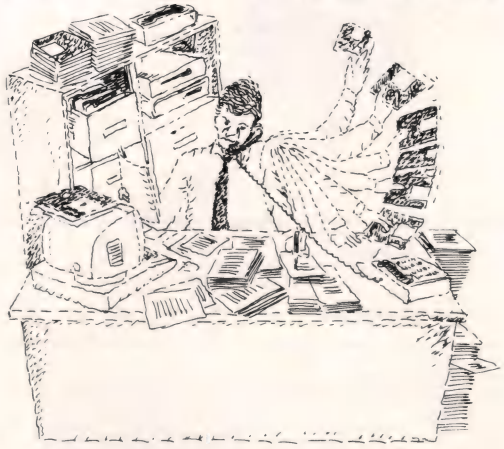


Figure 3: *Flames under an alien sun*

SOFTSTRIP®
NOW OFFERS YOU SOMETHING NEVER BEFORE AVAILABLE...



CONVENTIONAL DATA HANDLING

THE SOFTSTRIP SYSTEM

A CHOICE.

Until now you were stuck with disks.

No more. Install our unique STRIPPER™ software on your personal computer today and discover the many benefits of the fastest, easiest, least expensive way to handle information.

STRIPPER lets you print – ON PAPER – your own machine readable Softstrip data strips using your dot matrix printer. The Softstrip System Reader reads that information into a computer rapidly. With STRIPPER and the reader, your PC and printer become part of the most versatile information handling system available.

With this system you can do anything you wish with any data you have in your PC – ON PAPER.

DATA ENTRY: Why use keystrokes when you can eliminate them with data strips? Whatever the document - invoices, packing slips, memos, letters, sales reports, the list is endless — simply print a data strip right on the same printed page. Now you have a document that is both human readable and machine readable. A typical document can be entered in only 15 seconds using data strips. And, it ends keystroke errors forever.

DATA DISTRIBUTION: Why copy disks? It's time consuming and expensive. Softstrip data strips will end all that. Simply photocopy as many data strips as you like and send them by mail. Data strips ignore folding, coffee stains, ink marks and, by the way, magnetic fields. And if you're using telecommunications, you can stop making the phone company rich.

DATA STORAGE AND RETRIEVAL: Why have a file of disks and a file of paper? Eliminate one with Softstrip data strips. File the data strip with the document. Better still, print the strip right on the document. Then put it in a file or binder.

Retrieval is simple. To find existing data, pull the document and its related data strip from the file. They've been stored together. Then use the reader to enter the data. No more hassle trying to match documents with the right disk — if you can find it.

DATA TRANSFER: Why bother with cables, modems and phone lines to move files between computers? A Softstrip data strip generated by an IBM PC can be read into another PC, or compatible, an Apple or even a Macintosh. If you work at home on a Macintosh, make a data strip on your printer, take it into the office and read it into your IBM PC. Simple. And we've created the utilities to let you do that easily. (See Application Notes on opposite page.)

Fascinating, isn't it? Anything you can do with disks can be done with the Softstrip data strip system — faster, easier and at lower cost — ON PAPER.

All you need is STRIPPER software at **$19.95** and the Softstrip System Reader at **$199.95.**

A Digital Dissolve for Bit-Mapped Graphics Screens

by Mike Morton

How can one bit-mapped image fade to another?

As high-resolution, bit-mapped displays become more popular, computer screens are beginning to look like movie screens. Using fast bit-transfer subroutines, programs can pan, zoom, cut between images in the blink of an eye, and even animate in real time. All these effects use raw processor power to copy bit images as quickly as possible. But how about another standard film technique: dissolve shots? How can a program fade from one bit-mapped image to another? This article describes a way to do this very rapidly.

In the analog video world, fading from one image to another is easy: You just mix two images, bringing the new image's intensity up while decreasing the other's. In the more discrete software world, color or gray-scale displays can be dissolved by computing a weighted average of the old and new values for each pixel, then varying the weighting over time, much like with analog video.

Dissolving monochrome bit-mapped images is a different problem. You can't sum x percent of one pixel's bit and (100−x)percent of another's. There's too little variation per pixel to make a smooth transition from one image's pixel to another's. Although some displays have more than one bit of information per element, they still can't be faded by weighted averaging. For instance, a 24×80-character display has values that can't be averaged in the same way as color values or gray-scale levels can. (Can you imagine a screen that faded a 0 into a 9 by displaying all the digits in between in sequence?)

If monochrome bit maps (and some other displays) can't be averaged, how can they be dissolved? One solution is to copy pixels (or characters or whatever) to the screen in a more-or-less random order. This is easier said than done: unlike most problems involving random numbers, this one requires a random generator that produces each desired value exactly once.

Instead of producing a random sequence of coordinate pairs in an array of pixels, you can simplify the problem to a one-dimensional task by numbering each pixel starting with zero. If you can produce the pixel numbers in a random order, you can use that to copy the corresponding pixels to a screen in a random order and create a dissolve effect.

An analogous problem is a program to deal a deck of cards. If suits are numbered 0 to 3 and cards from 0 to 12 then, much like the pixel-numbering idea, you want to produce 52 unique coordinate pairs from (0,0) to (3,12). If you can produce a random sequence of integers from 0 to 51, then for each sequence element E, you can compute

```
suit  = E/13
card  = E mod 13
```

A common method for scrambling a set, such as the numbers 0 to 51, is to put the elements in an array and scramble the array by picking two random elements repeatedly and swapping them. This method has limits, though: How can you shuffle a million cards? To dissolve a 1,024×1,024-pixel monochrome bit map, you need the 1,048,576 pixel numbers scrambled. The storage demands of the array method are too much for many situations.

A Software Sequence Generator

The software magic that avoids using giant arrays is based on a simple hardware circuit. Figure 1, page 49, shows an 8-bit shift register. At each cycle, selected bits are sent through an N-way exclusive-OR gate, the entire contents of the register are shifted left by one bit, and the result of the exclusive-OR feeds in as the new rightmost bit. If the correct bit positions are fed into the exclusive-OR, the register will cycle through all possible nonzero combinations. If you interpret the register contents as a number, the sequence produces each of the num-

Mike Morton, c/o Lotus Development Corp., 161 First St., Cambridge, MA 02142

bers from 1 to 255 in a fairly random sequence. (How random is it? Not very, actually. It fails many statistical tests for randomness. But the circuit has a software analog that is easy to code and runs so fast that the imperfection can be overlooked.) For any size of register, you can exclusive-OR certain bit positions to make the register cycle through all nonzero combinations. A given size of register may have several correct patterns of bit positions; Table 1, page 50, shows one pattern for each register size from 2 to 32. (Also see inset article "Those Magic Constants" on page 55.)

An N-bit register produces the numbers from 1 through 2^N-1. My original goal was to produce the numbers from 0 through the size of the pixel array to be copied. I'll explain how to reconcile this difference as soon as I've turned this hardware sequence into software.

You could code a routine to mimic the circuit exactly, but there's a slightly different algorithm that is much faster. It uses a "mask" that corresponds to the bits selected in the circuit. For an 8-bit "register"—producing values from 1 to 255—one possible mask is 10111000, or $B8. Each sequence element is derived from the previous one with this method: Shift the original value right (not left) by one bit; if a "1" bit falls off the edge because of the shift, exclusive-OR the mask into the value. In assembly language, this can be done very easily. The following 68000 code takes a sequence element in *D0* and produces the next one in *D0* using the mask previously put in *D7*.

```
    LSR #1, D0      ; shift this sequence
                      element right 1 bit
    BCC.S noXOR     ; if no bit falls into
                      Carry, skip the
                      XOR . . .
    EOR D7,D0       ; . . . else XOR the
                      mask into the new
                      value
noXOR:              ; (now process the
                      new element in D0)
```

In C, it's not as easy to express the idea of a bit falling off the edge during a shift, so the code is more long-winded:

```
if (X&1)        /* is the low bit set? */
X = (X>>1)mask;
        /* yes: shift value and XOR in the
                                   mask */
else X = (X>>1);
        /* no: just shift the value */
```

A First Attempt

Now, with a software sequence generator and a way to map that sequence onto an array of pixels (or whatever), you can write a first cut at the dissolve algorithm. The idea is to find a "register width" for a sequence generator that generates at least as many numbers as there are pixels. (The highest-numbered sequence elements don't map to pixels in the array—that's OK, they'll be discarded.) Figure 2, below, illustrates this approach.

As the main loop generates each sequence element, the element is mapped to a pair of coordinates, just as in the card-dealing program. Coordinates outside the array are ignored, whereas those in bounds are copied. The loop ends when the sequence re-

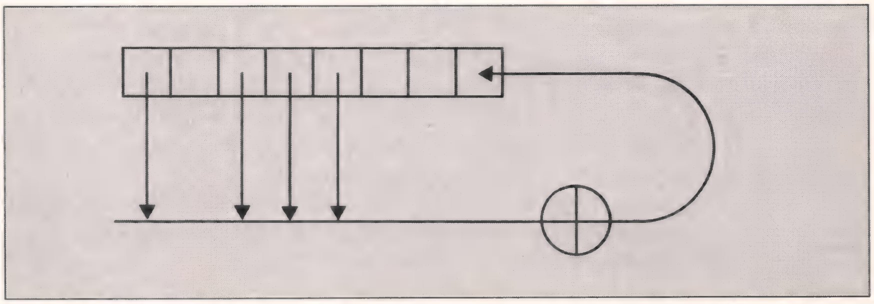


***Figure 1:** On 8-bit hardware sequence generator. The mask is 10111000 (base 2).*

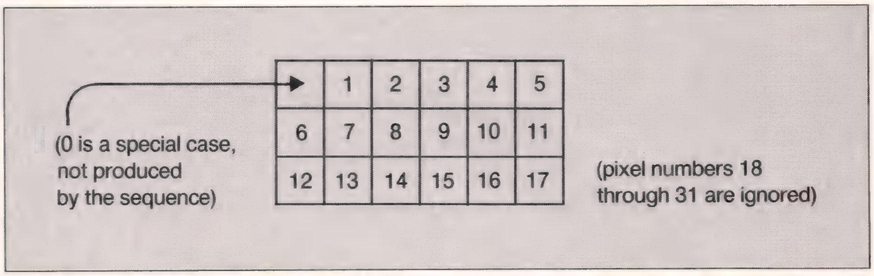


***Figure 2:** Mapping sequence elements into a pixel array using the formulas* row = N/width *and* column = N mod width. *Values of row >= height are ignored.*

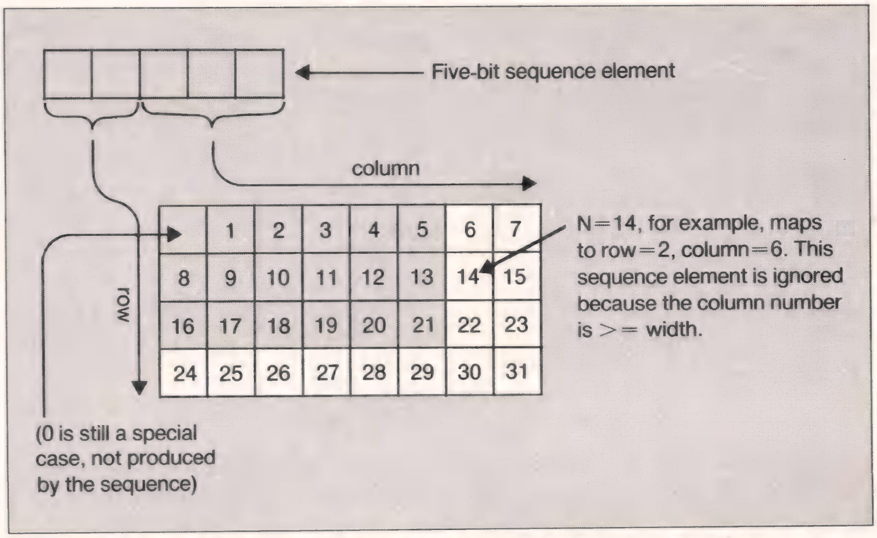


***Figure 3:** Mapping sequence elements into a pixel array using the formulas* row = N>>rowshift (rowshift = 3) *and* column = N&colmask *(colmask = 00111 base 2). Values of row >= height and values of column >= width are ignored.*

DIGITAL DISSOLVE
(continued from page 49)

turns to the original element. The function is shown in Table 2, page 52. This function finds the total number of pixels to copy and then the number of the highest pixel.

The correct width of sequence generator is found with a function called *bitwidth( )*. Given a number, this function just tells you how wide a register must be to hold the number. In other words: What width of generator is needed to produce at least as many pixel numbers as needed? The function *bitwidth* is shown in Table 3, page 54.

Once the width of the register has been found, the *randmasks[ ]* array is used to find the magic value used to generate the sequence. The sequence length can be nearly twice as long as the number of pixels to copy because the length must be a power of 2. The *[0]* and *[1]* elements of the array aren't defined—the smallest generator is 2 bits wide.

I haven't specified the *copy( )* routine—the routine that copies a pixel from one pixel array to another. It will depend on which computer you're using; I'll talk more about this later. Remember that *copy( )* doesn't need to copy a pixel to the screen; it can copy a character or any array element.

Because the sequence never produces a zero value, the program has to do the *(0,0)*th element explicitly. Eagle-eyed users will notice that the top-left pixel is always the first or last to dissolve in.

This method works, but it's not fast enough for many purposes. What makes it so slow is the division and modulo computations:

```
row = element / width;
column = element % width;
```

In assembly language, you can often do both of these with one instruction, but it's still awfully slow. A divide instruction on an 8-MHz 68000 takes about 17 microseconds. Is there a better way? (Of course, or I wouldn't have brought up the problem.)

A Faster Method

To make it faster, you can use a potentially longer sequence but use a mapping that is faster than the divide-and-modulo computation. Figure 3, page 49, illustrates this method. Both the height and width of the array are rounded up to the next power of 2. A sequence element is then broken up into row and column numbers by bit operations, which are much faster than divide and modulo. With this method, the number of sequence elements generated can be almost four times the number of pixels—twice as bad as the worst case for the simpler algorithm. But generating elements is so much faster than division that the new method is still faster. This algorithm is shown in Table 4, page 54. It is really a lot like the original routine. The difference is that the mapping is faster and the sequence may be longer because the mapping discards more elements of the sequence.

At one level, that's all there is to it. You just have to write a *copy( )* routine to copy a single pixel (or whatever) and optimize the code so the dissolve effect happens quickly. These tasks may not be simple for some machines, especially if you're trying to quickly copy tens of thousands or perhaps a million pixels on a high-resolution screen.

Every machine will probably need a slightly different *copy( )* routine to handle the quirks of your graphics hardware or software. In the Macintosh, for example, a bit map (such as the image to display or the screen to copy it to) is an array of pixels laid out in rows in memory. Indexing into it is done in just the same way as any linear array being treated as a two-dimensional array. Although most machines treat bit indexing differently from byte indexing, it may be helpful

Width	Mask (hex)	Produces from 1 to . . .
2	3	3
3	6	7
4	C	15
5	14	31
6	30	63
7	60	127
8	B8	255
9	110	511
10	240	1023
11	500	2047
12	CA0	4095
13	1B00	8191
14	3500	16383
15	6000	32767
16	B400	65535
17	12000	131071
18	20400	262143
19	72000	524287
20	90000	1048575
21	140000	2097151
22	300000	4194303
23	420000	8388607
24	D80000	16777215
25	1200000	33554431
26	3880000	67108863
27	7200000	134217727
28	9000000	268435455
29	14000000	536870911
30	32800000	1073741823
31	48000000	2147483647
32	A3000000	4294967295

For any given width (w), there is usually more than one mask that produces all values from 1 to $2^w - 1$. These particular masks are chosen because they can all be packed into a byte. See Listing One for an example of how to pack and unpack them.

Table 1: *Masks to produce pseudorandom sequences for registers from 2 to 32 bits wide*

A MEGABYTE FOR DOS!

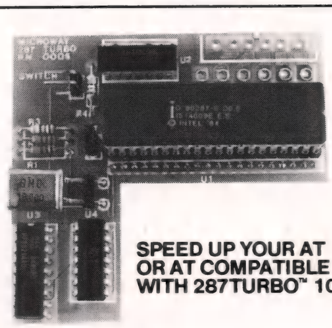
MicroWay is the world's leading retailer of 8087s and high performance PC upgrades. We stock a complete selection of 8087s that run from 5 to 12 MHz. All of our coprocessors are shipped with a diagnostic disk and the best warranty in the business – one year! We also offer daughterboards for socketless computers (NEC PC) and 287Turbo which increases the clock speed of the 80287 from 4 to 10 MHz. Our NUMBER SMASHER/ECM™ runs at 12 MHz with a megabyte of RAM and achieves a throughput of .1 megaflops with 87BASIC/INLINE, Intel Fortran, or Microsoft Fortran. Software reviewers consistently cite MicroWay software and 8087 expertise as the best in the industry! Our customers frequently write to thank us for recommending the correct software and hardware to meet their specific needs. They also thank us for our same-day shipping! In addition to our own products which support the 8087 and 80287, we stock the largest supply of specialized software available. For more information call us at

617-746-7341

NUMBER SMASHER/ECM™
THE FASTEST ACCELERATOR CARD AVAILABLE

gives you 12 MHz speed in two modes: 704K or one megabyte of "Extended Conventional Memory." MEGASWITCH MMU and MegaDOS software make it possible to run DOS applications with up to 1015K using PC compilers, AutoCAD and Lotus 1-2-3. Does not require EMS software. Totally compatible . . . from **$599** for 512K and 9.54 MHz to **$1199** with one megabyte and 8087-12.

Cooling Fan (for 12 MHz)........ **$80**



SPEED UP YOUR AT OR AT COMPATIBLE WITH 287TURBO™ 10 MHz

MicroWay® 8087 Support

For the IBM PC, PC XT, PC AT and Compatibles.

A2D-160™ MicroWay's Data Acquisition Board performs 160,000 12 bit Analog to Digital conversions per second! Includes software drivers. The fastest 12 bit A to D board available. For the IBM PC XT and compatibles.................. **$1295**

87SFL™ Accurate and fast **Scientific Subroutine Packages.** More than 145 functions: 34 Elementary Fcns, 18 Probability and Statistical Fcns, 15 Random Number Generators, Ei(x), $E_n(x)$, li(x), Si(x), Ci(x), $\Gamma(x)$, $\psi(x)$, $B(x,\omega)$, $I_x(a,b)$, erf x, S(x), C(x), $J_\nu(z)$, $Y_\nu(x)$, $I_\nu(z)$, $K_\nu(x)$, Ai(x), Bi(x), Ai′(x), Bi′(x), ber x, bei x, $\ker_1$ x, kei′ x, K(x), E(x), $F(\rho|a)$, $E(\rho|a)$, $\Pi(\rho|a,b)$, $\Lambda(a,b,\rho)$, $\mathcal{P}(z)$, $\mathcal{P}'(z)$, $P_n(x)$, $H_n(x)$, $L_n^{(\alpha)}(x)$, $J_n^{(\alpha,\beta)}(x)$, $G_n(p,q,x)$, $C_n^{(\alpha)}(x)$ and many more. Callable from FORTRAN, Pascal, BASIC and C. 1st module $250, 2nd on $100.

MATRIXPAK™ manages a **MEGABYTE!** Written in assembly language, our runtime package accurately manipulates large matrices at very fast speeds. Includes matrix inversion and the solution of simultaneous linear equations. Callable from RM or MS Fortran, MS Assembler, or 87BASIC/INLINE..................... each **$99**

87FFT™ Written in assembly language, performs Forward and Inverse FFTs on real and complex arrays which occupy up to 512 Kbytes of RAM. Also does convolutions, auto correlations, hamming, complex vector multiplication, and complex to radial conversions. Callable from most 8087 compatible compilers............... **$200**

87FFT-2™ performs two-dimensional FFTs. Ideal for image processing. Requires 87FFT **$100**

FASTBREAK™ 8087 support for Lotus 1-2-3™ Version 1A or 1A*................... **$79**

Microsoft Fortran V 3.31	**$209**
IBM Professional Fortran	**$565**
Ryan-McFarland Fortran V 2.0	**$399**
LAHEY Fortran F77L	**$477**
Grafmatic for Fortran or Pascal	**$125**
MultiHalo Graphics (1 language)	**$189**
Microsoft C V 4.0	**$299**
STSC APL★PLUS/PC	**$450**
INTEL, JRAM, AST, MAYNARD	**CALL**

MegaPage™ The only Intel-Lotus EMS board which comes with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache, and EMS drivers. For the IBM PC, XT & compatibles.... **$549**

DFixer™ Our disk utility which thoroughly checks PC or AT hard disks for bad sectors and updates the MS DOS file allocation table accordingly. Solves the AT hard disk problem! ... **$149**

DOptimizer™ Optimizes the way your hard disk or floppy stores its files. Speeds up accesses by recombining fragmented files............ **$49**

DCache™ Our disk caching software speeds up your I/O by storing repetitively used tracks in memory. The amount of memory used can be selected in 64 Kbyte banks................ **$49**

87MACRO/DEBUG™ Contains all the pieces needed for writing 8087/80287 assembly code & MicroWay's 87DEBUG debugger. **$199**

OBJ→ASM™ A multipass object module translator and disassembler. Produces assembly language listings which include public symbols, external symbols and labels commented with cross references. Ideal for patching object modules for which source is not available............ **$200**

87BASIC™ includes patches to the IBM BASIC or MS Quick BASIC Compiler for USER TRANSPARENT 8087 support. Provides super fast performance for all numeric operations including trigonometrics, transcendentals, addition, subtraction, multiplication, and division...... each **$150**

87BASIC/INLINE™ converts the output of the IBM BASIC Compiler into optimized 8087 inline code which executes up to seven times faster than 87BASIC. Supports separately compiled inline subroutines. Requires the IBM BASIC Compiler Version 1 and a Macro Assembler. Includes 87BASIC................................ **$200**

RTOS - REAL TIME OPERATING SYSTEM RTOS is a multi-user, multi-tasking real time operating system. It includes a configured version of Intel's iRMX-86, LINK-86, LOC-86, LIB-86, OH-86, and MicroWay's 87DEBUG. Runs on the IBM-PC, XT, PC-AT and COMPAQ................ **$600**

INTEL COMPILERS[1]

FORTRAN-86	**$750**
PASCAL-86	**$750**
PL/M-86	**$500**
ASM-86	**$200**

[1]Requires RTOS or iRMX-86. All Intel compiler names and iRMX-86 TM Intel Corp.

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and accurate installation instructions.

8087 5 MHz............... **$114**
For the IBM PC, XT and compatibles.

8087-2 8 MHz........... **$149**
For Wang, AT&T, DeskPro, NEC, Leading Edge.

80287-3 5 MHz......... **$179**
For the IBM PC AT and 286 compatibles.

80287-6 6 MHz......... **$229**
For 8 MHz AT compatibles.

80287-8 8 MHz......... **$295**
For the 8 MHz 80286 accelerator cards.

NEC V20, V30	**$16, $30**
64K RAM Set 150ns	**$10**
256K RAM Set 150ns	**$29**
256K RAM Set 120ns	**$39**
128K RAM Set PC AT	**$49**

287Turbo™ 10 MHz If you own an AT, Deskpro 286 or AT compatible, this is the card you need to get reasonable numeric performance. It plugs into your 80287 socket and includes a specially driven 10 MHz 80287. The card comes in three configurations. The IBM AT version includes a hardware RESET button................ **$450**

287Turbo 8 MHz.................. **$369**

87/88Turbo™ is a stubby card which includes a clock calendar and a speed controller which changes the speed of your motherboard from 4.77 to 7.4 MHz. Its use requires your PC to have a socketed 8284. Typical speed increase is 1.6 to 2.0. The card overcomes slow hardware by slowing up only when such devices are accessed and running at full speed otherwise........... **$149**
Optional 8087-2.................. **$149**

286TurboCache™ This accelerator for the PC or XT uses 8K of cache memory and 80286/80287 processors to provide an average speed increase of 3:1 for most programs. Call for specifications and benchmarks................ **$469**

Call for our complete catalog of software which supports the 8087.

In London, please phone 223-7662

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

```
dissolve1 (height, width)                 /* first version of the dissolve algorithm */
    int height, width;                    /* number of rows, columns */
{
    int pixels, lastnum;                  /* number of pixels; last pixel's number */
    int regwidth;                         /* "width" of sequence generator */
    register long mask;                   /* mask to XOR with to create sequence */
    register unsigned long element;       /* one element of random sequence */
    register int row, column;             /* row and column numbers for a pixel */

    /* Find smallest register that produces enough pixel numbers */
    pixels = height * width;              /* compute number of pixels to dissolve */
    lastnum = pixels-1;                   /* find last element (they go 0 . . lastnum) */
    regwidth = bitwidth (lastnum);        /* how wide must the register be? */
    mask = randmasks [regwidth];          /* which mask is for that width? */

    /* Now cycle through all sequence elements. */
    element = 1;                          /* 1st element (could be any nonzero) */
    do {
        row = element / width;            /* how many rows down is this pixel? */
        column = element % width;         /* and how many columns across? */
        if (row < height)                 /* does this seq element fall in the array? */
        copy (row, column);               /* yes: copy the (r,c)th pixel */

        /* Compute the next sequence element */
        if (element & 1)                  /* is the low bit set? */
        element =(element >>1) mask;      /* yes: shift value, XOR in mask */
        else element = (element >>1);     /* no: just shift the value */
    } while (element != 1);               /* loop until we return to original element */

    copy (0, 0);                          /* kludge: the loop doesn't produce (0,0) */
}                                         /* end of dissolve1( ) */
```

***Table 2:** First attempt at dissolve algorithm*

DIGITAL DISSOLVE

(continued from page 50)

to keep all offsets as bits and convert to bytes only when necessary. Thus, on the Mac, if *D0* contains the bit offset of the first bit of the array in memory, and *D1* and *D2* have the row and column numbers, you can set the *(D1,D2)*th bit with the code fragment shown in Table 5, page 54.

All the arithmetic in Table 5 is in terms of bits until the *LSR* extracts the address, which then has to be moved to an address register to be useful. Bit numbering in the Mac screen isn't like byte numbering (byte numbers increase as you move to the right across the screen; bit numbers decrease within each byte), so the bit number in *D3* has to be converted with the *NOT*. Finally, the *BSET* sets the bit in the correct byte. This idea of mapping a row and column in the array to an address in memory is the heart of any *copy( )* routine for a bit map. You use it to test the bit in the bit map being dissolved in and to set or clear the corresponding bit in the screen memory. For a 24×80-character display, it's similar: the mapping takes the row and column and indexes into a character array in memory to get the character to copy to the screen.

The Macintosh Dissolve Routine

Listing One, page 88, is a dissolve routine for the Mac. Because its calling interface resembles that of the standard Mac bit-transfer routine called CopyBits, it's called DissBits. It copies the contents of one rectangle in a bit map to another rectangle in another bit map. In the Mac's graphics subsystem, a bit map is a data structure that specifies the base address of a chunk of graphical data and imposes a coordinate system on that chunk. For more information, see the chapter on QuickDraw in *Inside Macintosh* by Caroline Rose et al.

DissBits is not intended to present a clear example of the algorithm; instead, the main loop contains the *copy( )* code in-line for maximum speed. DissBits actually has three different loops and automatically uses the fastest one it can. The first loop is the general case; with some coding tricks, it can dissolve at the rate of

Circle no. 233 on reader service card.

```
int bitwidth (N)         /* find "bit-width" needed to represent N */
    unsigned int N;      /* number to compute the width of */
{
    int width = 0;       /* initially, no bits needed to represent N */

    while (N != 0) {     /* loop 'til N has been whittled down to 0 */
        N >>= 1;         /* shift N right 1 bit (N.B.: N is unsigned) */
        width++;         /* and remember how wide N is */
    }                    /* end of loop shrinking N down to nothing */

    return (width);      /* return bit positions counted */
}                        /* end of bitwidth( ) */
```

Table 3: *The function* bitwidth

```
dissolve2 (height, width)                /* fast version of the dissolve algorithm */
    int height, width;                   /* number of rows, columns */
{
    int rwidth, cwidth;                  /* bit width for rows, for columns */
    int regwidth;                        /* "width" of sequence generator */
    register long mask;                  /* mask to XOR with to create sequence */
    register int rowshift;               /* shift distance to get row from element */
    register int colmask;                /* mask to extract column from element */
    register unsigned long element;      /* one element of random sequence */
    register int row, column;            /* row and column for one pixel */

    /* Find the mask to produce all rows and columns. */
    rwidth = bitwidth (height);          /* how many bits needed for height? */
    cwidth = bitwidth (width);           /* how many bits needed for width? */
    regwidth = rwidth + cwidth;          /* how wide must the register be? */
    mask = randmasks [regwidth];         /* which mask is for that width? */

    /* Find values to extract row and col numbers from each element. */
    rowshift = cwidth;                   /* find dist to shift to get top bits (row) */
    colmask = (1<<cwidth)-1;             /* find mask to extract bottom bits (col) */

    /* Now cycle through all sequence elements. */
    element = 1;                         /* 1st element (could be any nonzero) */
    do {
        row = element >> rowshift;       /* find row number for this pixel */
        column = element & colmask;      /* and how many columns across? */
        if ((row < height)               /* does element fall in the array? */
            && (column < width))         /* . . . must check row AND column */
        copy (row, column);              /* in bounds: copy the (r,c)th pixel */

            /* Compute the next sequence element */
        if (element & 1)                 /* is the low bit set? */
        element = (element >>1) mask;    /* yes: shift value, XOR in mask */
        else element = (element >>1);    /* no: just shift the value */
    } while (element != 1);              /* loop until we return to original element */

    copy (0, 0);                         /* kludge: element never comes up zero */
}                                        /* end of dissolve2( ) */
```

Table 4: *A faster dissolve algorithm*

```
MULU     <WIDTH>,D1    ; convert row number to bit offset of row
ADD.L    D1,D0         ; compute bit offset of first bit of the row
ADD.L    D2,D0         ; index into the row to get the bit offset
MOVE.L   D0,D3         ; set aside the final bit offset . . .
LSR.L    #3,D0         ; . . . and find the byte address from it
MOVE.L   D0,A0         ; copy that to an address register
NOT.B    D3            ; convert the bit index to 68000-style
BSET     D3,(A0)       ; set the D3th bit of A0's byte
```

Table 5: *Code fragment to set* (D1,D2)*th bit on the Mac*

DIGITAL DISSOLVE
(continued from page 52)

about 49 microseconds per pixel. When both the bit maps have a width that is a power of 2, then the *MULU* instruction in Table 5 can be done with a shift instead. In this case, the algorithm uses a different loop that is about 20 percent faster. As a third and final optimization, the loop-choosing code notices when the pixel array being copied is exactly the width of the bit map it's contained in. (This case is common because it includes doing a full-screen dissolve.) In this case, the check for the column being in bounds can be removed and some other tricks can be done. In this "high gear," the code runs at 19 microseconds per pixel; it can fade the whole Mac screen in about 3.4 seconds.

Suggestions

Those of you writing for other machines may not want to wade through the details of the Macintosh code. Besides the optimizations above, here are some things you may want to try:

- All my code uses 32-bit long words; this slows it down considerably but is needed for cases in which there are more then 2^{16} bits to dissolve. If your machine has fewer than 64K pixels, you can always use 16-bit integers. Or you can code two loops and have the routine choose the right one.
- You can optimize when either the width or the height of the array is a power of 2. In this case, you know the column or row (respectively) extracted from the sequence element is always in bounds.
- You can test the upper bits (the row, as I've split things up) of the sequence element without extracting them first. Just compare it to 1 plus the maximum allowed value, shifted into position.
- In assembly language, it's easy to detect when the loop has finished if the last time through the loop the sequence element is 1. When you shift the 1 right, you'll get a carry and the result will be 0. This carry-and-zero occurs only if the element was 1. (On some machines, a single instruction will detect when both of these conditions are true—for instance, the 68000's *BLS* (branch less than)

instruction.)

- When there's no carry, the mask isn't XORed into the next sequence element, and the topmost bit of the new element is 0. This means that the row (in my version) is guaranteed to be in bounds. Your branch-on-no-carry can take you to the start of the loop as usual but bypass the check for the row being in bounds.
- Because both the row and column must be checked to see if they're in bounds, extract and check one before taking the time to extract the other.
- The row number is extracted by shifting the sequence element to the right, but this result is soon multiplied by the width of the bit map when finding the bit. If the width is a multiple of a power of 2, part of the multiply is just undoing some of the effect of the shift. Because longer shifts take more time on some processors, you can reduce both the shift and the amount to multiply by—if you are willing to mask the value, too.

There are some nagging, real-world problems. A few I had trouble with were:

- Don't forget the *(0,0)*th element of the array. Because the sequence never produces the value 0, this won't get done by the loop. You have to make it a special case outside the loop.
- The algorithm breaks down for tiny bit maps because the sequence generator doesn't work for small widths of registers. If you can find a way to generalize it, I'd like to hear about it. (My solution was to detect this case and hand it off to the CopyBits routine.)
- Your machine's graphics software will probably not be able to copy thousands of pixels quickly one at a time, and your *copy( )* routine will have to be in assembly language and directly alter screen memory. Be careful to check exactly how this interferes with the graphics software. For instance, DissBits must hide the Mac cursor during the time it's directly accessing screen memory.

There are a lot of variations I haven't tried yet; I'd be interested to hear from anyone who experiments with these ideas:

- My *copy( )* routine for bit maps tests

```
; flip: A function to return a random boolean value in D0.b.
; D7 is used to keep the current state of the "shift register".
; The register is 16 bits, so the pattern of bits repeats after 64K calls.

flip:   LSR.W    #1, D7        ; shift D7 right one bit
        SCS      D0            ; set D0 to flag whether a bit carried
        BCC.S    end           ; if no bit carried, D7 is new element: done
        EORI.W   #$B400,D7     ; else XOR in magic value for new one
end:    RTS                    ; return with D0.b = random boolean
```

Table 6: *Sequence generator for coin-flip subroutine*

Random Bits vs. Random Bytes

The shift-and-XOR method the dissolve uses to generate pseudorandom numbers isn't very random. If you watch some sizes of rectangles being dissolved, you'll see ephemeral patterns. Surprisingly, though, the pattern of bits shifted off the end of the register is random. In *The Art of Computer Programming*, Knuth discusses the difference between random bits and numbers composed of the bits.

Many programmers writing games or other programs wind up flipping a coin with something like the BASIC statement *IF RND < 0.5 THEN. . . .* This is a lot of effort to produce a random bit. You can write a short, fast coin-flip subroutine using the sequence generator in Table 6, above. Remember to initialize *D7* to any nonzero value. Zero traps the subroutine in a demon state, producing nothing but zero bits.

Here's a puzzle: Change the function so it doesn't need *D7* initialized. You can write a "self-starting" version that is no slower and no longer than the original.

Those Magic Constants

When the sequence generator shifts a 1 bit out of the register, it exclusive-ORs a magic constant into the running value of the register. Different register widths require different constants; Table 1 gives a list of some constants that work. How do you find these values?

The basis for the generator is rooted in the theory of prime polynomials. Those of you who remember your abstract algebra course may be interested in pages 209—213 of *Numerical Recipes: The Art of Scientific Computing* by William H. Press et al. and pages 29—31 of *The Art of Computer Programming* by Donald E. Knuth.

If you prefer direct evidence, it's not hard to write a program to search for the constants. I set a microcomputer to work on the problem, using the slower method of directly simulating the type of circuit shown in Figure 1 (I didn't know about the software version at that point). It took about two CPU-weeks.

Assuming I haven't made transcription errors in Table 1, you shouldn't have much reason to search for the constants yourself. (Perhaps some constants are more random than others?) The table should give you plenty of information—at least until someone produces billion-pixel displays. But just in case, Press et al. give constants for registers up to 100 bits wide.

DIGITAL DISSOLVE
(continued from page 55)

a bit in the source and then branches to either set or clear a bit in the destination. If, before the dissolve, the destination is exclusive-ORed into the source, the source becomes a bit map of the bits that differ between the two images. Then the *copy( )* routine still has to test the source but toggles the destination bit only when the source bit is 1—this means the address calculation for the destination's bit can often be skipped. If your graphics subsystem supports logical operations (such as exclusive-OR) on block copies, the first XOR pass can be very fast. Finally, if you do a third block-XOR from the destination back to the source, the net effect of the three XORs is to swap the two images, so your destination is preserved in the source and can be restored easily.

- Because the loop's state is controlled by one variable (the sequence element), it should be easy to write a partial dissolve by changing the starting or ending elements of the sequence. This would allow you to "blend" images, create effects like the Star Trek transporter, and so forth.
- Because all this came from a hardware circuit, how about hardware support for the dissolve? Many machines have custom hardware to support raster operations; why not support this one?
- Instead of copying graphical data, variations on the dissolve theme could perform standard Boolean operations with two or more images, invert an image, and so on.

Finally, don't let your imagination be limited to dissolve subroutines. The random engine used in the dissolve has other uses. You can do much more than shuffle a deck of a million cards. For instance, a program slowly drawing the Mandelbrot set could draw pixels randomly so that the basic image becomes apparent sooner, without having to wait for the entire drawing. In browsing through an image database with a slow modem, you can cancel an image without waiting for it to be completely drawn (I believe this idea is Ted Nelson's). As described in the inset article "Random Bits vs. Random Bytes" on page 55, you can write a coin-flip subroutine. The unusual property of generating unique values in pseudorandom order makes the sequence generator a useful tool for many applications.

Bibliography

Knuth, Donald E. *The Art of Computer Programming*. vol. 2. 2d ed. Reading, Mass.: Addison-Wesley, 1981. 29–31.

Press, William H., et al. *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press, 1986. 209–213.

Rose, Caroline, et al. *Inside Macintosh*. vol. 1. Reading, Mass.: Addison-Wesley, 1985. Chapter 6.

DDJ

(Listing begins on page 88.)

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

C DYNAMO™

FREE! UNIX† WORKALIKE FREE!

(See Limited Time Special Offer Below)

PAINT TEXT, HELP, MENU & DATA ENTRY SCREENS & WINDOWS FROM KEYBOARD • COMPLETE INPUT FORMATTING • FULL VALIDATION • UNLIMITED WINDOWS • AUTOMATIC WINDOW OVERLAY & RESTORE • FULL ATTRIBUTE CONTROL & HIGHLIGHTING • 1-2-3 LIKE MENUS • DRAW BOXES & BORDERS • GENERATE "C" CODE WITH TOUCH OF A KEY • LEFT/RIGHT/CENTER JUSTIFICATION • SPECIFY ANY FILL CHARACTER & CURRENCY SYMBOL • U.S./EUROPEAN NUMBER STYLES • MORE DATA TYPES • ONE CALL WILL OPERATE ENTIRE SYSTEM • SUPPORTS IBM PC/XT/AT & COMPATIBLES • SUPPORTS MICROSOFT, COMPILER LATTICE, AZTEC, CI86, ANY FULL K & R OR ANSII COMPILER

NEW DYNAMO DATA ENTRY

UNIQUE, POWERFUL, NECESSARY

Full Validation of Each Keystroke
Range Checking
Data Security to Item Level
Scrollable Data Entry Forms with Fixed & Scrollable Parts
Allows Forms Larger Than Screen

Over 34 Item Types

Powerful "Picture" Capability
Unique: Mix Text, Data Entry Fields With Menu Items

Full source code. No Royalties
Code plus manual *$129.95*

POWER WINDOWS

MOST POWERFUL YET
POP-UP WINDOWS FOR

Menus/Overlays
Help Screens
Messages/Alarms

ZAP ON/OFF SCREEN
FILE-WINDOW MANAGEMENT
COMPLETE CONTROL OF:

Cursor
Attributes
Borders

AUTOMATIC

Horizontal & Vertical Scrolling
Word Wrap
Line Insertion

The most powerful, flexible and easy to use windowing package available! Many types of menus. Highlighting. Move data between files, keyboard, program and windows. Status lines. Change size/location/overlapping. Move/add/delete/cascade windows. Full source code. No royalties.
3 disks *$129.95*

C FUNCTION LIBRARY

BEST YOU CAN GET
325 FUNCTIONS
SUPERB DOCUMENTATION

Most complete screen handling plus graphicscursrsor/keyboard/data entry, 72 string functions with word wrap; status andcontrol; utility/DOS BIOS/ time/date functions; printer control & more. Special functions. Full source code. No royalties.
4 disks *$129.95*

†TM AT&T

NEW DYNAMO SCREEN PAINTER AND FORMS CREATOR

DOES IT ALL
RIGHT FROM YOUR KEYBOARD
AUTOMATIC CODE GENERATION

Data or Help Screens & Windows
Data Entry Screens & Windows
Menus

FAST, FLEXIBLE, EASY

Save Man-Months of Programming
Full Control of Screen Attributes
Monochrome or Color
by Form, Screen, Window or Item

FAST EASY MENU GENERATION

1-2-3 Like, Many Others
Full Read/Write Security by Item

Requires Dynamo Data Entry
Screen painter & manual *$129.95*

B-TREE LIBRARY & ISAM DRIVER

POWERFUL DATA MANAGER

Fixed/Variable length records
Fast! Easy To Use!
16.7 Million Records/File
16.7 Million Keys/File

Full source. No royalties. *$129.95*
***MAKE** Utility (Snake)* *$59.95*

SUPERFONTS FOR C

Super Size Characters
Monochrome adapter
Color/graphics adapter
8 Font Libraries

Font and Function Library *$49.95*

C-TERP

INTERPRETER FOR C

No Compromise, Full K&R
Built In Screen Editor
Fast, Fast Compile/Link
Use External Libraries!
Symbolic Debugging
Single Stepping
Rave Reviews!

2 disks and manual *$299.95*

COMBINE AND SAVE!

SCREEN PAINTER + DATA ENTRY
BOTH for only $179.95
(A $260 VALUE)

C LIBRARY plus POWER WINDOWS
BOTH for only $179.95
+ SUPERFONTS FOR C $199.95
(A $310 VALUE)

C BUSINESS LIBRARY
INCLUDES C FUNCTION LIBRARY, POWER WINDOWS, SUPERFONTS FOR C, B-TREE LIBRARY, ISAM
ALL for..................$299.95
(A $440 VALUE + FREE UNIX WORKALIKE)

C TOTAL LIBRARY
INCLUDES C FUNCTION LIBRARY, POWER WINDOWS, SUPERFONTS FOR C, B-TREE LIBRARY, ISAM and C-TERP C interpreter
ALL for..................$499.95
(A $740 VALUE + FREE UNIX WORKALIKE)

C DYNAMO LIBRARY #1
SCREEN PAINTER, DYNAMO DATA ENTRY, POWER WINDOWS, C FUNCTION LIBRARY, SUPERFONTS for C
ALL for..................$249.95
(A $570 VALUE)

C DYNAMO LIBRARY #2
ALL OF C DYNAMO LIBRARY #1
PLUS B-TREE & ISAM $349.95
(A $700 VALUE + FREE UNIX WORKALIKE)

C DYNAMO LIBRARY #3
ALL OF C DYNAMO LIBRARY #2
PLUS C-TERP $549.95
(A $1000 VALUE + FREE UNIX WORKALIKE)

**FREE PC UNIX, PCVMS OR O/S TOOL BOX WITH ANY PURCHASE OVER $295.00 LIMIT ONE PER CUSTOMER
Offer expires 11/30/86

SPECIAL OFFER

OPERATING SYSTEMS

Multi-User Multi-Tasking Networking, Full Source

	LIST	
PCNX™ UNIX WORKALIKE	*$99*	*FREE**
PCVMS™	*99*	*FREE**
O/S TOOL BOX™ (Build your own operating system)	*99*	*FREE**

™WENDIN SOFTWARE
*with purchase of any Entelekon combination library priced at $295 or more.
Limit 1 Special Offer item per customer.
Offer expires 11/30/86

THE C DYNAMO FAMILY...C TOOLS THAT WORK *TOGETHER*

Entelekon™

SINCE 1982

12118 Kimberley, Houston, TX 77024   713-468-4412   VISA-MASTERCARD-CHECK-COD

Listing One (Text begins on page 14.)

```
                         Listing 1 -- set.h
-------------------------------------------------------------------------
/* SET.H:        Header to use the set manipulation routiness in set.c
 *
 * Copyright (c) 1986, Allen I. Holub.  All rights reserved
 */

#define DEFBYTES        2               /* bytes in def set (>= 1)      */
#define DEFBITS         (DEFBYTES * 8)  /* bits in default set          */

typedef struct
{
        unsigned        nbytes : 13;     /* Number of bytes in map      */
        unsigned        compl  : 1;      /* This is a negative true set */
        int             nbits;           /* Number of bits in map       */
        unsigned char   *map;            /* Pointer to the map          */
        unsigned char   defmap[DEFBYTES]; /* The map itself             */
}
SET;

extern  void    set_op  ( int,  SET*, SET*, SET*); /* Routines in set.c */
extern  int     set_cmp ( SET*, SET*    );
extern  int     subset  ( SET*, SET*    );
extern  int     num_ele ( SET*          );
extern  void    delset  ( SET*          );
extern  SET     *newset (               );

#define UNION           0       /* x is in s1 or s2                     */
#define INTERSECT       1       /* x is in s1 and s2                    */
#define DIFFERENCE      2       /* x is in s1 but !s2 or in s2 but !s1  */
#define INVERT          3       /* ones complement                      */
#define ASSIGN          4       /* s1 = s2                              */
#define CLEAR           5       /* d = all bits cleared                 */
#define FILL            6       /* d = all bits set                     */

#define union(d,s1,s2)          set_op( UNION,      d,     s1,     s2   )
#define intersection(d,s1,s2)   set_op( INTERSECT,  d,     s1,     s2   )
#define difference(d,s1,s2)     set_op( DIFFERENCE, d,     s1,     s2   )
#define assign(d,s1)            set_op( ASSIGN,     d,     s1,     NULL )
#define invert(d,s1)            set_op( INVERT,     d,     s1,     NULL )
#define clear(d)                set_op( CLEAR,      d,     NULL,   NULL )
#define fill(d)                 set_op( FILL,       d,     NULL,   NULL )
#define complement(d)           ( (d)->compl = ~(d)->compl)

#define equivalent(s1,s2)       ( set_cmp(s1,s2) == 0 )
#define disjoint(s1,s2)         ( set_cmp(s1,s2) == 1 )

#define GBIT(x,s,op)    ( ((s)->map)[(x) >> 3]  op  (1 << ((x) & 0x07))    )

#define remove(x,s)     ( ((x) >= (s)->nbits) ? 0           : GBIT(x,s,&= ~) )
#define add(x,s)        ( ((x) >= (s)->nbits) ? addset(x,s) : GBIT(x,s,|=  ) )
#define ismember(x,s)   ( ((x) >= (s)->nbits) ?  0          : GBIT(x,s,&   ) )

#define test(x,s)       ( ( ismember(x,s) ) ? !( (s)->compl ) : (s)->compl  )
```

End Listing One

Listing Two

```
                    Listing 2 -- set.c
------------------------------------------------------------------------------
#include  <stdio.h>
#include  <ctype.h>
#include  <set.h>

extern  char    *calloc ( int, int );

/*----------------------------------------------------------------------*/

#ifdef DIAG
#       define D(x) x
#else
#       define D(x)
#endif

#define max(a,b)        ((a) > (b) ? (a) : (b))

/*----------------------------------------------------------------------*/

SET     *newset()
{
        SET     *p;

        if( !(p=(SET *) calloc(sizeof(SET),1)) )
        {
                fprintf(stderr,"Can't get memory for set\n");
                return NULL;
        }
        else
        {
                p->map    = p->defmap;
                p->nbytes = DEFBYTES;
                p->nbits  = DEFBITS;
        }
        return p;
}

/*----------------------------------------------------------------------*/

void    delset( set )
SET     *set;
{
        /* Delete a set created with a previous newset
        */

        if( set->map != set->defmap )
                free( set->map );

        free( set );
}

/*----------------------------------------------------------------------*/

static  void    enlarge( need, set )
SET     *set;
{
        /*      Enlarge the set to "need" bytes, filling in the extra
         *      bytes with zeros.
```

(continued on next page)

Listing Two **(Listing continued, text begins on page 14.)**

```
         */

        register char   *new;

        if( !set  ||  need <= set->nbytes )
                return;

        D( printf("enlarging %d byte map to %d bytes\n", set->nbytes,need);)

        if( !(new = calloc(need, 1)) )
                fprintf(stderr,"Can't get memory to expand set\n");
        else
        {
                memcpy( new, set->map, set->nbytes );

                if( set->map != set->defmap )
                        free( set->map );

                set->map    = new;
                set->nbytes = need;
                set->nbits  = need * 8;
        }
}

/* ------------------------------------------------------------------- */

int     addset( bit, set )
SET     *set;
{
        /*      Addset is called by the add() macro when the set isn't
         *      big enough. It expands the set to the necessary size
         *      and sets the indicated bit.
         */

        enlarge( (bit >> 3) + 1, set );
        GBIT    (  bit, set, |= );
}

/* ------------------------------------------------------------------- */

int     num_ele( set )
SET     *set;
{
        /*      Return the number of elements (set bits) in the set.
         *      This routine depends on zero fill when an
         *      unsigned quantity is shifted to the right.
         */

        register unsigned j;
        register unsigned count = 0;
        unsigned char     *p;
        int               i;

        p = set->map;
        for( i = set->nbytes; --i >= 0 ; p++)
                for( j = *p;  j  ;  j >>= 1 )
                        count += j & 0x1;

        return count;
}

/* ------------------------------------------------------------------- */

set_cmp( set1, set2 )
SET     *set1, *set2;
{
        /* Compares two sets. Returns zero if they're equivalent, one if
         * they're disjoint, 2 if they intersect but aren't equivalent,
         * -1 is returned if the two sets are different sizes.
         */

        register char   *p1, *p2;
        register int    i, disj = 0 ;

        i = max( set1->nbytes, set2->nbytes);

        enlarge( i, set1 );             /* Make the sets the same size  */
        enlarge( i, set2 );

        p1 = set1->map;
        p2 = set2->map;

        for(; --i >= 0; p1++, p2++ )
        {
                if( *p1 != *p2 )
                {
                        if( *p1 ^ ~*p2 )
                                return 2;
                        else
                                disj = 1;
                }
        }
```

(continued on page 62)

Listing Two (Listing continued, text begins on page 14.)

```

        return disj;            /* They're equivalent   */
}

/* ------------------------------------------------------------------ */

int     subset( a, b )
SET     *a, *b;
{
        /*      Return 1 if A is a subset of B. Set A must be either smaller
         *      than or the same size as B. 0 is returned if A is not a
         *      subset or if A is larger than B.
         */

        register int    i;
        register char   *ap, *bp;

        if( (i = a->nbytes) > b->nbytes )
                return 0;

        ap = a->map;
        bp = b->map;

        for(; --i >= 0; ap++, bp++ )
                if( (*ap & *bp) != *ap )
                        return 0;
        return 1;
}

/* ------------------------------------------------------------------ */

void    set_op( op, dest, set1, set2 )
int     op;
SET     *set1, *set2, *dest;
{
        /*      Performs either the union or intersection of two sets
         *      (depending on the value of "union"). Dest is the result.
         *      The two source sets (set1 and set2) must be different,
         *      however either of the sources may be used as a destination
         *      if you like. If the sets are different sizes, the smaller
         *      set is made larger. Unused arguments should be set to NULL.
         */

        register char   *d;             /* Pointer to destiniation map  */
        register char   *m1;            /* Pointer to map in set1       */
        register char   *m2;            /* Pointer to map in set2       */
        register int    i;              /* Number of bytes in map       */


        i = dest->nbytes;
        if( set1 )
                i = max( i, set1->nbytes );
        if( set2 )
                i = max( i, set2->nbytes );

        enlarge( i, set1 );     /* Make all three sets the same size    */
        enlarge( i, set2 );     /* if necessary. Enlarge() does nothing */
        enlarge( i, dest );     /* if they're already the correct size. */

        d  = dest->map;
        m1 = set1->map;
        m2 = set2->map;

        while( --i >= 0 )
        {
                D( printf("set_op: working on bit %d\n", i ); )

                switch( op )
                {
                case UNION:        *d++ =  *m1++ | *m2++        ; break;
                case INTERSECT:    *d++ =  *m1++ & *m2++        ; break;
                case DIFFERENCE:   *d++ =  *m1++ ^ *m2++        ; break;
                case ASSIGN:       *d++ =  *m1++                ; break;
                case INVERT:       *d++ =  ~*m1++               ; break;
                case CLEAR:        *d++ =  0                    ; break;
                case FILL:         *d++ = ~0                    ; break;
                }
        }
}

/* ------------------------------------------------------------------ */
#ifdef DEBUG

pset( str, set )
char    *str;
SET     *set;
{
        int     i;

        printf("+-----------------------------------------------------\n");
        printf("| %s\n", str );
        printf("| Set at 0x%04x: %d bits, %d bytes, map (0x%04x) ",
```

(continued on page 64)

NOW AT THE SBC MART
COMPUTING SALE-A-THON

1.2MB Floppy on your PC or XT

★ **Now you can have a 1.2MB-and-360K *AT-compatible* floppy disk drive on your IBM (or compatible) PC or XT. Our 1.2MB controller replaces your floppy disk controller. The controller is $149, a TEAC 1.2MB/360K disk drive is $135, the driver is $36. The controller combined with clock/calendar, serial and parallel ports, and able to control 3.5″ and 8″ floppy disk drives is only $189.**

SPECIAL BONUS Norton Utilities *Ver 3.0* Reg $99 **only $60** w/purchase of any disk drive. IBM PC-DOS 3.2 (the real thing) Reg $95 **only $70,** w/purchase of motherboard.

ADD-ON CARDS

- multifunction card 384K, clk/cal w/battey, serial, parallel, and game ports, with 0K save $50 **Model MFC-4 $89**
- monochrome graphics card runs 1-2-3 graphics, w/printer port, 720x348, **MGC-1 now $99** (Hercules equiv.)
- floppy controller for 1-4 48tpi *or 96tpi* drives, w/cable save 125 **Model FDC-2 now only $59**
- color adapter w/light pen port, RGB and composite outputs Reg $140 **Model CC-1 now just $79**

★ **better than the Super 7:** floppy disk controller, clk/calendar, serial, parallel & game ports, spooler & RAMDISK s/w **Sale $99** ideal mate for 640K motherboards

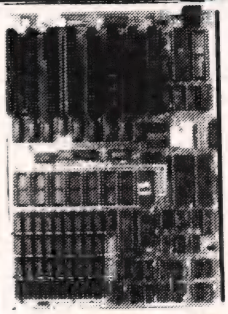
- clock/calendar card Reg $58, **CL-1 $49**

★ 808K ON A SINGLE FLOPPY, TEAC 96tpi 80-track disk drive kit, complete, nothing else to buy **now $169**

SBC PLEDGE

★ service after sale
★ low prices
★ technical support
★ one year warranty

MOTHERBOARDS

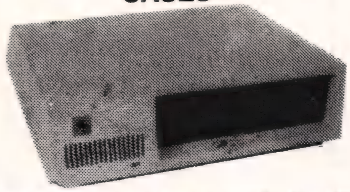


— full IBM PC/XT compatibility
— 8 I/O slots
— runs IBM's PC-DOS 3.1 & 3.2
— BASIC interpreter available
— great foundation for business or personal system
— 1 year warranty

★ **Model MB-2 640K Turbo** motherboard 4.77 & **true 8MHz** clock (switchable from keyboard or by software) with 256K RAM Reg $349 **now $239**

- **Model MB-3 640K** motherboard with 0K Reg $199 **now $159**
- **TURBO Program** see below, right.

CASES



— high quality IBM PC look alike w/sideswitch, flip-flop top or side chassis

— any-combo disk drive brackets

— heavy steel 5- and 8-cutout style

- **Model CA-8** Reg $95, **now $69**

ABOUT OUR MOTHERBOARDS

— the most IBM PC compatible motherboards available

— each runs *all* commercially available software

— each works with all commerciallly available add-on cards

These FOUNDATION motherboards are the most compatible you can buy. All work with V20 chips, have parity checked memory, an 8087 socket, 8 adapter slots, four empty ROM sockets. Super manual includes complete data on how to put together a complete PC system. It even includes schematics.

HARD DISKS

— complete 20 MB kit

— **special only $479**

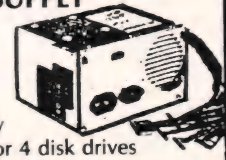
- complete internal ½-high 20MB kit, incl. controller and cables, **special $479**
- complete 30 MB kit with ½-high drive, uses new Adaptec 2070-A controller, **$579**

★ OMTI hard disk controller card (this card is super fast) Reg $220, **Model HDC-1 now $175,** w/cables

★ Generalized hard-disk controller works with almost any drive. Just tell the controller your drive specs. Fast OMTI 5510-7 hard disk controller with cables and OMTIDISK utilities **HDC-4 only $189**

- "Assembling an IBM PC/XT Compatible Computer System" A novice can put an IBM PC compatible together with this new detailed manual. **$19**
- Microsoft's new QuickBASIC full MS BASIC compiler, Reg $99, **only $89**

POWER SUPPLY



— 135 Watts
— side switch
— top quality
— one year warranty
— standard cables for 4 disk drives

- **Model PS-135** power supply Reg $129, **now only $75**

TURBO PROGRAM

★ This program will set virtually all turbo motherboards to the turbo, or high-speed, mode. Can be called from the keyboard or from your AUTOTEXEC file **only $19**

BEST DOS MANUAL

★ Microsoft Press' "Running MS DOS". For the less experienced and the most knowledgeable MS/PC DOS user. 423 pages. We regularly get calls from our customers telling us how great this book is. **$21.95**

Shipping and handling: drives/motherboards/cases/power supplies **$4.50 ea.,** cards **$2 ea.,** software **$2.50 ea.,** keyboards/modems **$4 ea.,** speedup kits **$2 ea.,** memory **$1/set,** COMPUTERFACTS **$3 first set then $1/set.** CA residents add 6% sales tax.

master charge

VISA

Visa/MC/AmEx ORDERS:(619) 375-5744
The SBC MART, P.O. BOX 1296, Ridgecrest, CA 93555

AMERICAN EXPRESS

The SBC Mart is part of Computing Technology, 247 Balsam Street, Ridgecrest, CA 93555

Circle **no. 183** on reader service card.

Listing Two (Listing continued, text begins on page 14.)

```
                                    set, set->nbits, set->nbytes, set->map);

        printf("%s TRUE\n", set->compl ? "NEGATIVE" : "POSITIVE" );
        printf("| map = ");

        for( i = 0; i < set->nbytes; i++ )
                printf("0x%02x,", (set->map)[i] );

        printf("\n| bits= ");

        for( i = 0; i < set->nbits; i++ )
                printf( test(i, set) ? "%d," : "" ,  i );

        printf("\n| %d elements\n", num_ele(set) );

        printf("+-----------------------------------------------------\n");
}

/* -------------------------------------------------------------------- */

test_stuff( a, b, d )
SET *a, *b, *d;
{
        pset("set a", a );
        pset("set b", b );

        union        (d,a,b);   pset("a union b",       d );
        intersection (d,a,b);   pset("a intersect b",   d );
        difference   (d,a,b);   pset("a difference b",  d );
        assign       (d,a);     pset("d assign a",      d );
        complement   (d);       pset("complement a",    d );
        complement   (d);
        invert       (d,a);     pset("invert a",        d );

        printf("a %s equivalent to b\n", equivalent(a,b) ? "IS" : "ISN'T" );
        printf("a %s disjoint from b\n", disjoint(a,b)   ? "IS" : "ISN'T" );
        printf("b %s a subset of a\n",   subset(b, a)    ? "IS" : "ISN'T" );
        printf("a %s a subset of b\n",   subset(a, b)    ? "IS" : "ISN'T" );

        printf("-----------------------------------------------------\n");
}

/* -------------------------------------------------------------------- */

main()
{
        SET     *a, *b, *d;
        char    buf[80], *p;
        int     num;

        a = newset();  pset( "initial a", a );
        b = newset();  pset( "initial b", b );
        d = newset();  pset( "initial d", d );

        add(0,a);
        add(1,a);
        add(3,a);
        add(0,b);
        add(3,b);

        test_stuff( a, b, d );

        remove(0,a); remove(1,a); remove(3,a); remove(0,b); remove(3,b);
        add(0, a); add(2, a); add(2, b); add(3, b);

        test_stuff( a, b, d );

        clear( a ); clear( b ); test_stuff( a, b, d );
        clear( a ); fill  ( b ); test_stuff( a, b, d );

        delset( b );
        delset( d );
        delset( a );
        a = newset();

        printf("enter <bitnum><s|c>:");

        while( gets(buf) )
        {
                num = atoi(buf);
                for( p = buf; isdigit(*p) ; p++ )
                        ;

                if( *p == 's' )
                        add(num ,a);
                else
                        remove(num, a);

                pset( "", a );
                printf("enter <bitnum><s|c>:");
        }
}

#endif
```

End Listings

Listing One (Text begins on page 30.)

```
Draw a rectangle with an 8086 on CGA.


;
;       Draw a rectangle in the upper
;       left corner of a CGA display
;       in high-resolution mode.  The code
;       is hardwired to a 10x10 rectangle.
;
;
; Set up segment and offset registers
; to point to display memory.
;
        mov         AX, 0B800H
        mov         ES, AX
        mov         BX, 0
;
; Draw the top line by stuffing one byte
; and the first two bits of the next byte.
;
        mov         byte ptr [BX], 0FFH
        mov         byte ptr [BX+1], 0C0H
;
; Draw the bottom line the same way.
;
        mov         byte ptr [BX+800], 0FFH
        mov         byte ptr [BX+801], 0C0H

;
; Draw the first and last pixels on the
; next 4 even scan lines, then do the same
; on the odd scan lines.
;
        mov         SI, 50H
        mov         CX, 4
EOLoop: mov         byte ptr [BX+SI], 80H
        mov         byte ptr [BX+SI+1], 40H
        add         SI, 80
        loop        EOLoop
        cmp         SI, 2000H
        jg          EODone
        mov         SI, 2050H
        mov         CX, 4
        jmp         EOLoop
EODone  label       byte
;
; Rectangle is finished.
;

*******************************************

   Same rectangle drawn by 34010


;
; Draw a line from 0,0 to 0,10.  The start
; point is in register B2 and the end point
; (delta X and delta Y) is in register B7.
;
; The > sign precedes a 32-bit hex constant.
;
        MOVI        >0,B2
        MOVI        >00100000,B7
        LINE        0
;
; Repeat the process for the other sides.
;
        MOVI        >00100000,B2
        MOVI        >00000010,B7
        LINE        0
        MOVI        >0,B2
        MOVI        >00000010,B7
        LINE        0
        MOVI        >00000010,B2
        MOVI        >00100000,B7
        LINE        0
;
; Finished!
;
*******************************************


   Same rectangle drawn by 82786


;
; Move to the upper left corner and
; draw a 10x10 rectangle.
;
        ABS_MOVE  0,0
        RECT      10,10
;
; All finished!
;
```

End Listing

Explore AI on your PC

Smalltalk/V transforms your PC into a versatile AI workstation

Only Smalltalk/V lets you experience the thrill of a responsive AI workstation while learning artificial intelligence techniques and using them to create practical applications.

"Smalltalk/V gives me the feel of an AI workstation on my PC."
—Darryl Rubin, Technical Editor, AI Expert Magazine

Watching someone use an AI workstation is a vision of what the computer was meant to be. Fingers dance across the keys as windows dilate, shift, overlap, and disappear on the bit-mapped display. Ideas spring to life as program fragments execute, are modified, expanded, combined and tried again in a creative arabesque of text and graphics. The interface vanishes, man and machine are one. Smalltalk/V brings that experience to your IBM-PC.

"We use Smalltalk as our primary language for teaching artificial intelligence."
—Dr. John Pugh, Director, School of Computer Science, Carleton University

Of the three main AI languages, Smalltalk, LISP, and Prolog, only Smalltalk was intended for individual use on a personal computer. Only Smalltalk was designed to provide a match between human and computer models of reality. Only Smalltalk is easy to learn, easy to read, and easy to use.

"We found Smalltalk/V excellent for developing advanced decision-support tools based on decision analysis and AI techniques."
—Dr. Samuel Holtzman, Professor, Stanford University

Smalltalk/V is pure object-oriented programming — a powerful tool for designing frame/script-based knowlege representations, inference engines, expert systems, simulation environments, intelligent interfaces, network control software, communications interfaces, and much more.

Methods, our character-based Smalltalk, is now available for $79. It has all of the features of Smalltalk/V except graphics, rules, source-level debugger, and object-swapping. However, it supports color, includes the communication package, and does not require a mouse.

BYTE and BIX are trademarks of McGraw-Hill, Inc. IBM, IBM-PC, and IBM PC-AT are trademarks of International Business Machines Corporation. Unix is a trademark of Bell Laboratories.

"Smalltalk/V is the highest performance object-oriented programming system available for PCs."
—Dr. Piero Scaruffi, Chief Scientist, Olivetti Artificial Intelligence Center

$99.00

Smalltalk/V Features

- High-performance object-oriented programming
- Integrates object-based and rule-based programming with object-oriented Prolog
- A user-extensible, open-ended environment
- A responsive graphical user interface
- Supports exploratory programming and prototyping
- Class hierarchy with inheritance creates highly re-useable code
- Smalltalk source code included, with browser windows for easy access and modification
- A huge toolkit of classes and objects for building a variety of applications
- Object-swapping creates virtual memory on hard or RAM disk
- Bit-mapped graphics with bit and form editors
- A sophisticated source-level debugger
- Automatic change log for easy recovery from errors
- Powerful directory/file browser system for organizing DOS files
- Optional communications interface to Unix™ and other systems
- Access to other languages and DOS functions
- DOS command shell
- Detailed owner's manual designed for both beginners and advanced programmers

YES! I want to turn my PC into an AI workstation. Send me . . .

- ☐ Smalltalk/V $99
- ☐ Communications for Smalltalk/V . . . $49
- ☐ Methods (including Communications) $79

Shipping and Handling .... $______

CA residents add applicable sales tax .... $______

TOTAL .... $______

Shipping and Handling
U.S., Canada, and Mexico .... $ 5.00
Elsewhere $15.00

I enclose ☐ Check ☐ Money Order

☐ Credit card information ☐ MC ☐ VISA

Number: ______________ Expiration____

Signature ______________

Name: ______________

Street Address: ______________

City/State/Zip: ______________

Phone: ______________

NOT COPY PROTECTED, 60-DAY MONEY-BACK GUARANTEE
ON-LINE USER-SUPPORT CONFERENCE ON BYTE'S BIX™

Smalltalk/V requires DOS and 512K RAM on IBM PCs (including AT) or "compatibles," a Microsoft or compatible mouse, and a CGA, EGA, Hercules, or AT&T Hi-Res graphic controller.

digitalk inc.

5200 West Century Boulevard
Los Angeles, CA 90045 (213) 645-1082

Circle **no. 127** on reader service card.

PC Network: the world's #1 supplier o

*members pay wholesale, + 8% and get 14-30 day software rentals

*PC Network members pay just 8% above the wholesale price, plus shiping. All prices reflect a 3% cash discount. Wholesale prices fluctuate rapidly—all prices in this issue have been prepared 60 days in advance. Please call for latest prices. Minimum shipping $2.50 per order. International orders call for shipping & handling charges. Money Order, personal and company checks please allow 10 working days to clear.

Arity Corporation

Expert System Development Package	$235.00*
File Interchange Toolkit	37.00*
Prolog Compiler and Interpreter	650.00*
Prolog Interpreter	275.00*
SQL Development Package	235.00*
Screen Design Toolkit	37.00*
Standard Prolog	70.00*
Arity Combo Package	950.00*

PC NETWORK MEMBERSHIP APPLICATION

YES! Please enroll me as a member in the PC NETWORK™ and send my catalog featuring thousands of computer products, all at just 8% above DEALER WHOLESALE PRICES. I will also receive "THE PRINTOUT", a special update on merchandise at prices BELOW even those in my wholesale catalog and all the other exclusive, money-saving services available to Members. I am under no obligation to buy anything. My complete satisfaction is guaranteed.

Please (✔) all boxes that apply:

•Basic Membership With 14 Days Rental — 1 Year $8 ☐ 2 Year $15 ☐
Business Software Rental Library $25 ☐ add'l. per year
Games Software Rental Library $10 ☐ add'l. per year

•Special V.I.P Membership With 30 Days Rental BOTH Business and Game Software Rental Libraries — 1 Year $15 ☐ 2 Year $25 ☐
$30 ☐ add'l. per year

☐Bill My Credit Card: ☐VISA ☐MasterCard ☐American Express
Account Number: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Exp. Date ________ mon. yr.

☐Check or Money Order Enclosed for $ ________

Name ________

Address ________ Apt. No. ____

City ________ State ________ Zip ____

Telephone ( ) ________

My Computer(s) is: ☐IBM PC ☐IBM XT ☐IBM AT ☐Apple II ☐Macintosh Other ____

Signature ________
(Signature required to validate membership)

Blaise Computing Inc.

Asynch Manager for C	$115.00*
Asynch manager for Pascal	115.00*
C Tools/Tools 2 Combo Pack	115.00*
C Tools	85.00*
C Tools	65.00*
C Tools Plus	115.00*
Exec	65.00*
Pascal Tools/Tools 2 Combo	115.00*
Pascal Tools	85.00*
Pascal Tools 2	65.00*
Runoff	35.00*
Turbo Asynch Plus	65.00*
Turbo Power Tools Plus	65.00*
View Manager w/Source for C	185.00*
View Manager w/Source for Pascal	185.00*

Borland International

Reflex	$54.00*
Reflex and Workshop Bundle	99.95*
Reflex Workshop	35.25*
Turbo Database Toolbox	34.00*
Turbo Editor Toolbox	35.00*
Turbo Gameworks Toolbox	35.00*
Turbo Graphix Toolbox	32.97*
Turbo Lightning	48.50*
Turbo Pascal w/8087 and BCD Support	50.00*
Turbo Tutor	19.50*
Word Wizard	37.25*
Word Wizard and Turbo Lightning Bundle	79.50*

Lattice

C-Compiler 3.1	$250.00*
C-Sprite Debugger	115.00*
C-Food Smorgasbord	82.00*
Sidetalk	79.00*
DBC-III Library	137.00*
RPG-II Compiler	410.00*

LogiTech

LogiMouse	$75.00*
LogiMouse +	89.00*
LogiMouse + w/PC Paint	117.00*
LogiMouse + w/Generic CAD	139.00*
LogiMouse + w/Reflex	145.00*

CALL TOLL FREE
1-800-621-SAVE

(Orders —Membership and Advice!)
In Illinois call (312) 280-0002
Your Membership Validation Number DD21
You can validate your membership number and, if you wish, place your first money-saving order over the phone by using your VISA, MASTERCARD or AMERICAN EXPRESS. **Our knowledgeable sales consultants are on duty Mon.-Fri. 8:00 AM to 7:00 PM, Sat. 9:00 AM to 5:00 PM CST.**

PERSONAL COMPUTER NETWORK
320 West Ohio Street
Chicago, Illinois 60610

Call now...Join the PC NETWORK and start saving today!
Customer Service and Order Status (312) 280-1567
8:30 AM to 4:30 PM, Mon.-Fri. CST

WHY BUY OUR $159 UNIX SYSTEM FOR YOUR PC/AT WHEN YOU CAN GET THE SAME THING FOR A THOUSAND DOLLARS MORE?

HERE ARE THE REASONS WHY.

Best-price Unix applications software. From the company that made a reality of best-price UNIX operating systems.

Business computing, word processing, database management, text preparation and software development—we've selected, developed and added value to the best. Once you've converted your PC/AT, with our System V, into a true multi-user, multi-tasking machine, you'll demand software that lets you get on with the tasks at hand.

CRYSTALWRITER PLUS

is a comprehensive word-processing system for the multi-user office. It allows concurrent use of files, layouts and printers, as well as supporting a full range of terminals.

INFORMIX SQL

is the relational-database-management system developers will choose for custom applications. It offers relational-database flexibility, interactive-database queries, a flexible design environment, automated functional menus, custom screen design and total format control.

R-OFFICE

ties a powerful word processor together with the file-management, table-spreadsheet, report-generation and desktop-management chores common to nearly every modern office.

R/M COBOL

is Ryan-McFarland's GSA certified implementation of the ANSI X3.23 74 standard. R/M COBOL is designed for serious business computing.

LEX

is a standalone system offering complete word processing capabilities, as well as utilities for adapting LEX to any type of video terminal or printer.

SOFTWARE DEVELOPMENT

from Microport. Our package was derived from the extensively-tested Portable C compiler. (The large-model code produced by the compiler is among the densest and fastest available.) These are true 286 compilers, making full and efficient use of the extended 286 instruction set. The package also includes the F77 Fortran compiler.

TEXT PREPARATION

from Microport. This package consists of the complete System V, Release 2 Documenter's Workbench (DWB), and includes both the new troff and the old troff. Drivers for the HP Laser Jet Printer and Apple's LaserWriter are also available.

These are but a few of our available packages. Virtually everything listed in AT&T's catalog can now, or will soon run with our operating system.

Regarding Our SYSTEM V operating system.

More than a look-alike, Microport's System V complies with AT&T's new binary standard and was derived from AT&T's *own* UNIX System V, Release 2 iAPX286. It thereby contains standard System V features the competitors don't support, such as the powerful symbolic debugger (SDB), the shell-layering job-control facility and the F77 Fortran compiler, as well as programming tools such as ctrace, cflow and bs. Over 200 utilities come standard to let you search and modify files, make use of electronic mail, calculate electronically, convert data and publish.

- **24 hour sale and support**
- **90 day money-back guarantee**
- **Educational site licensing**
- **Commitment to AT&T souce and binary standards**

To order: Call either of our numbers below for today's price and product line quote (be sure to ask about our buy-two-applications-packages-and-get-RUNTIME-free offer).

You really wouldn't want to pay thousands of dollars more, would you?

M I C R O P O R T

MICROPORT SYSTEMS, INC.
4200 Scotts Valley Drive
Scotts Valley, CA 95066
408/438-UNIX (CA) or 800/PC2-UNIX

"In the marketplace, the ultimate wisdom belongs to the customer."

(Bidwell's Law; *Good Reading)*

"I have been buying software from about six mail order houses in the U.S. and your response time has been the best that I have encountered."

Rodolfo S. Suaco
Kowloon, Hong Kong

"In any company, it is a great rarity to find courteous and timely service, product knowledge objectively offered, more than reasonable prices, and a willingness to pursue the latest new market offerings. My experience strongly indicates Programmer's Connection considers this their minimum acceptable standard in meeting customer needs."

Craig Timko
Car Programs Management
Ford Motor Company

"I'm overwhelmed by how you bend over backwards to get things for us!"

Karen Stevens
Mindscape Software Inc.
San Jose, California

"I am delighted!...your service is excellent. This is my first dealing with Programmer's Connection, but not the last, I am sure."

Ken Travers
KenT Computer Services
Hornsby Heights, Australia

"Excellent service, excellent selection."

Reported by **Mark Ackerman**
For the Boston Computer Society *PC Report.*

Turn the page for our product listing and ordering information.

QUALITY

Quality products and quality people. That's what Programmer's Connection is all about.

We carry the finest selection of the best programmer's development tools specifically for IBM Personal Computers and compatibles. They are the latest versions and most come with 30-day return guarantees or evaluation periods.

We firmly believe that high quality must be present throughout every aspect of our service. And to make sure that we maintain such high standards, we include a service questionnaire with every purchase. We're very interested in what our customers have to say.

SUPPORT

Our courteous, knowledgeable, non-commissioned salespeople are always ready to assist you. We also have experienced technical consultants on staff who can answer questions about products and provide sound, unbiased advice.

We'll support you before and after you make your purchase. Your satisfaction is very important to us.

PRICE

Our buying power enables us to offer you the lowest prices without sacrificing service.

UPS Ground shipping is FREE to all U.S. customers. There are no extra charges for credit cards, CODs, purchase orders or special handling (except for export preparation).

Quite simply, the prices on the next two pages are all you pay.

INTEGRITY

When we started Programmer's Connection in 1984, we dedicated ourselves to providing high quality personal service to every customer. Since then, we've quickly grown to be the leading independent dealer in this industry.

We're very proud of the trust we've earned from our customers and we pledge always to be worthy of it.

It's our commitment to quality, support, low prices and integrity that sets us apart. So make the connection today and find out for yourself what our customers are talking about.

PROGRAMMER'S CONNECTION
Service
QUALITY INTEGRITY SUPPORT PRICE

PROGRAMMER'S CONNECTION
QUALITY · INTEGRITY · Service · SUPPORT · PRICE

apl language

	LIST	OURS
APL*PLUS/PC *by STSC* ... *New Version*	595	449
APL*PLUS/PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS/PC Tools Vol 1 *by STSC*	295	199
APL*PLUS/PC Tools Vol 2 *by STSC*	85	59
APL*PLUS/UNX *For AT XENIX by STSC*	995	695
Btrieve *ISAM File Mgr by SoftCraft*	250	194
Financial/Statistical Library *by STSC*	275	195
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	795	599

arity products

Arity Combination Package ... *New*	1225	1139
Arity PROLOG Compiler & Interpreter	795	699
Arity Expert System Development Pkg	295	259
Arity File Interchange Toolkit	50	45
Arity Screen Design Toolkit	50	45
Arity SQL Development Package	295	259
Arity PROLOG Interpreter	350	309
Arity Standard Prolog	95	85

artificial intelligence

APES *by Prog Logic Sys*	395	339
APT *from Solution Systems* ... *New*	65	CALL
AutoIntelligence *by IntelligenceWare* ... *New*	990	CALL
ESP ADVISOR *by Expert Systems Intl* ... *New*	895	839
PROLOG-2 Interface ... *New*	395	369
ExpertEDGE Advanced *by Human Edge* ... *New*	2500	CALL
ExpertEDGE Professional *by Human Edge* ... *New*	5000	CALL
Experteach II *by IntelligenceWare*	475	379
EXSYS *Development Software by EXSYS*	395	319
First Class *by Human Edge*	495	399
GCLISP *Golden Common LISP by Gold Hill*	495	CALL
GCLISP 286 Developer *by Gold Hill*	1190	CALL
Insight 1 *by Level Five Research*	95	75
Insight 2+ *by Level Five Research*	485	379
Intelligence/Compiler *IntelligenceWare*	990	749
Logic-Line Series 1 *by Thunderstone*	90	85
Logic-Line Series 2 *by Thunderstone*	125	115
Logic-Line Series 3 *by Thunderstone*	150	139
LPA microPROLOG *by Prog Logic Sys*	250	219
with APES	450	399
LPA Professional microPROLOG	395	339
with APES	695	599
Microsoft LISP *Common LISP*	250	175
PC Scheme *by Texas Instruments* ... *New*	95	85
Personal Consultant Easy *by TI* ... *New*	495	439
Personal Consultant Plus *by TI* ... *New*	2950	2599
Personal Consultant Runtime ... *New*	CALL	CALL
PROLOG-2 Interpreter *by ESI* ... *New*	450	419
PROLOG-2 Interpreter and Compiler ... *New*	895	839
QNIAL *by NIAL Systems*	375	349
TransLISP *from Solution Systems* ... *New*	95	CALL
Turbo PROLOG Compiler *by Borland Intl*	100	75

assembly language

386 ASM/LINK *Cross Asm by Phar Lap* ... *New*	495	CALL
8088 Assembler *by 2500 AD*	100	89
ASMLIB *Function Library by BC Assoc*	149	129
asmTREE *B-Tree Dev System by BC Assoc* ... *New*	149	129
Cross Assemblers *Various 2500 AD*	CALL	CALL
Microsoft Macro Assembler	150	98
Turbo EDITASM — by Speedware	99	84
Visible Computer: 8088 *Software Masters*	80	65

basic language

BetterBASIC *by Summit Software*	200	139
8087 Math Support	99	79
Btrieve Interface	99	79
C Interface	99	79
Run-time Module	250	219
Microsoft QuickBASIC ... *New Version*	99	75
Professional BASIC *by Morgan*	99	75
8087 Math Support	50	42
True Basic *with BASICA Converter* ... *New*	200	105
True Basic *w/Converter & Run-time* ... *New*	350	199
Advanced String Library ... *New*	50	45
Asynch Communication Support	50	45
BASICA Converter	50	45
Btrieve Interface	50	45
Developer's Toolkit	50	45
Formlib	50	45
Hercules Graphic Support ... *New*	50	45
Run-time Module	150	109
Sorting & Searching ... *New*	50	45

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	135
C TOOLS	125	99
C TOOLS 2	100	79
C TOOLS PLUS	175	135
EXEC *Program Chainer*	95	75
PASCAL TOOLS	125	99
PASCAL TOOLS 2	100	79
PASCAL TOOLS & PASCAL TOOLS 2	175	135
RUNOFF *Text Formatter*	50	45
TURBO ASYNCH PLUS	100	83
TURBO POWER TOOLS PLUS	100	83
VIEW MANAGER *Specify C or Pascal*	275	199

borland products

REFLEX *Data Base System*	150	99
REFLEX Workshop	70	55
REFLEX & REFLEX Workshop	200	149
Turbo DATABASE TOOLBOX	70	50
Turbo EDITOR TOOLBOX	70	50
Turbo GAMEWORKS TOOLBOX	70	50
Turbo GRAPHIX TOOLBOX	70	50
Turbo LIGHTNING	100	74
Turbo PASCAL *with 8087 and BCD*	100	69
Turbo Prolog Compiler	100	75
Turbo TUTOR *for Turbo PASCAL*	40	28
Word Wizard	70	50
Word Wizard and Turbo Lightning	150	119

c++

C++ *from Guidelines* ... *New*	195	179

c compilers

C-86 *by Computer Innovations*	395	279
Datalight C Compiler *Small Model*	60	49
Datalight Developer Kit *w/Large Model*	99	79
DeSmet C *w/Debugger*	159	145
DeSmet C *w/Debugger & Large Case*	209	193
Eco-C *Development System by Ecosoft*	125	89
Lattice C Compiler *from Lattice*	500	294
Mark Williams Let's C	75	58
with csd *Source Debugger*	150	109
Mark Williams MWC-86	495	289
Microsoft C *with CodeView*	450	288
Wizard C Combo *by Wizard Systems* ... *New*	750	599
Wizard C Compiler	450	359
ROM Development Pkg ... *New*	350	299

c interpreters

C-terp *by Gimpel, Specify compiler*	300	235
C Trainer *by Catalytix* ... *New*	99	CALL
Instant C *by Rational Systems*	500	CALL
Introducing C *by Computer Innovations*	125	104
Run/C *from Lifeboat*	150	89
Run/C Professional *from Lifeboat*	250	169

c utilities

APT *by Shaw American Technology*	395	299
Basic C Library *by C Source*	175	129
C Essentials *by Essential Software*	100	CALL
C-ISAM *by Relational Database Sys* ... *New*	225	195
C to dBase *by Computer Innovations*	150	135
c-tree *ISAM File Manager by FairCom*	395	329
r-tree *Report Generator* ... *New*	CALL	CALL
C Utility Library *Essential* ... *New Version*	185	135
C Windows *by Syscom*	100	89
C Wings *by Syscom*	50	45
CI Probe *by Computer Innovations*	225	189
CI ROMPac *by Computer Innovations*	195	139
dbQUERY *by Raima*	195	155
dbVISTA *Single-User DBMS by Raima*	195	155
with Source Code	495	425
dbVISTA *Multi-User DBMS by Raima*	495	425
with Source Code	990	845
dBx *dBase/C Translator by Desktop AI*	350	325
Entelekon Combo Package	200	169
C Function Library	130	109
C Windows	130	109
Superfonts for C	50	43
Essential Graphics *by Essential Software*	250	205
Flash-up Windows *by Software Bottling*	75	68
GraphiC *Mono v2.2 by Sci Endeavors*	280	209
GraphiC *Color v3.0 by Sci Endeavors*	350	289
GRAFLIB *by The Librarian* ... *New*	175	CALL
Greenleaf Comm Library *by Greenleaf*	185	134
Greenleaf Data Windows *by Greenleaf* ... *New*	225	189
with Source Code ... *New*	450	379
Greenleaf Functions *by Greenleaf*	185	134
The HAMMER *by OES Systems*	195	149
HALO *by Media Cybernetics*	300	209
HELP/Control *by MDS* ... *New*	125	109
MetaWINDOWS *No Royalties*	185	115
MetaFONTS	80	CALL
MetaWINDOWS/Plus *by Metagraphics*	235	189
MetaFONTS/Plus	235	CALL
On-line Help *from Opt-Tech Data Proc*	149	109
PANEL *by Roundhill Computer Systems*	295	224
PC Lint *by Gimpel Software*	139	105
PLOTHI *by The Librarian* ... *New*	175	CALL
PLOTHP *by The Librarian* ... *New*	175	CALL
Sci Subroutine Library *by Peerless*	175	139
Vitamin C *by Creative Programming*	150	135
VC Screen *Forms Designer*	100	84
Zview *by Data Management Consultants*	245	189

cobol language

Micro Focus COBOL Workbench	4000	3379
Micro Focus Level II COBOL	1500	549
COGRAPHICS	250	199
COMATH	200	159
FORMS-2	300	259
Level II Animator	900	349
Level II SOURCEWRITER	2000	CALL
Micro Focus Level II COBOL *for Novell*	2000	1699
Micro Focus Micro/SPF	175	149
Micro Focus Professional COBOL	3000	2295
Multi-user Runtime *for PC Network*	500	429
Microsoft COBOL	700	445
Microsoft COBOL Tools *w/Debugger*	350	205
Realia COBOL ... *New Version*	995	785
RM/COBOL *by Ryan-McFarland*	950	639
RM/COBOL 8X *ANSI 85 by Ryan-McFarland*	1250	895

debuggers & profilers

386 DEBUG *Cross Debugger by Phar Lap* ... *New*	175	139
Advanced Trace-86 *by Morgan Computing*	175	138
CI Probe *by Computer Innovations*	225	189
Codesifter *Profiler by David Smith*	119	98
Codesmith-86 *by Visual Age*	145	108
DSD86 *by Soft Advances* ... *New*	70	65
DSD87 *by Soft Advances* ... *New*	100	89
Periscope I *by Data Base Decisions*	295	245
Periscope II *w/NMI Breakout Switch*	145	109
Periscope II-X *Software only*	115	84
The PROFILER *with Source Code by DWB*	125	94
The WATCHER *Profiler by Stony Brook* ... *New*	60	55

forth language

CFORTH *Native Code Compiler by LMI*	300	239
Forth/83 Metacompiler *Specify Target*	750	599
PC/Forth *by Laboratory Microsystems*	150	119
PC/Forth+ *by Laboratory Microsystems*	250	209
Advanced Color Graphics Support	100	79
Enhanced Graphics Support	200	159
Intel 8087 Support	100	79
Interactive Symbolic Debugger	100	79
Native Code Optimizer	200	159
PCTERM *Modem Pgm for Smartmodem*	100	79
Software Floating Point	100	79

fortran language

50 MORE: FORTRAN *by Peerless Engr*	125	99
ACS Time Series *by Alpha Computer*	495	419
Btrieve *ISAM File Mgr by SoftCraft*	250	194
For-Winds *by Alpha Computer Service*	90	78
Forlib-Plus *by Alpha Computer Service*	70	54
FORTLIB *by The Librarian* ... *New*	95	CALL
FORTRAN Addenda *by Impulse Engr*	95	89
FORTRAN Addendum *by Impulse Engr*	165	149
GRAFLIB *by The Librarian* ... *New*	175	CALL
HALO *by Media Cybernetics*	300	209
I/O PRO *with NO LIMIT Library by MEF* ... *New*	390	349
Microcompatibles Combo Package	240	219
Grafmatic	135	119
Plotmatic	135	119
Microsoft FORTRAN	350	208
NO LIMIT *by MEF Environmental* ... *New*	129	115
PANEL *Screen Designer by Roundhill*	295	224
PLOTHI *by The Librarian* ... *New*	175	CALL
PLOTHP *by The Librarian* ... *New*	175	CALL
RM/FORTRAN *by Ryan-McFarland*	595	389
Sci Subroutine Library *by Peerless*	175	138
Sci Subroutine Package *by Alpha*	295	249
Strings & Things *by Alpha Computer*	70	54

gss products

GSS Graphics Development Toolkit ... *New*	495	CALL
GSS Kernel System *for DOS* ... *New*	495	CALL
GSS Kernel System *for IBM RT* ... *New*	795	649
GSS Metafile Interpreter ... *New*	295	239
GSS Plotting System ... *New*	495	CALL
GSS SOLUTIONS Chart ... *New*	295	239
GSS SOLUTIONS Plottalk ... *New*	295	239
GSS SOLUTIONS Terminal ... *New*	295	239

lattice products

Lattice C Compiler *from Lattice*	500	294
with Library Source Code	900	545
C Cross Reference Generator	50	39
with Source Code	200	149
C-Food Smorgasbord *Function Library*	150	98
with Source Code	300	188
C-Sprite *Source Level Debugger*	175	138
Curses *Screen Manager*	125	94
with Source Code	250	184
dBC *dBase File Manager for C*	250	189
with Source Code	500	378
LMK *Make Facility*	195	145
RPG II Compiler *No Royalties*	750	635
SecretDisk *File Encryption Utility*	120	94
SideTalk *Resident Communications*	120	94
Text Management Utilities	120	94
TopView Toolbasket *Function Library*	250	189
with Source Code	500	378
Z-80 C Cross Compiler	500	378
with Library Source Code	1000	749

Circle **no. 129** on reader service card.

logitech products

LOGIMOUSE C7 *Mouse Hardware*	99	85
with PLUS Pkg	119	99
with PLUS Pkg & PC Paintbrush	169	145
with PLUS Pkg & CAD Software ... New	189	159
with PLUS Pkg & CAD & Paint ... New	219	189
MODULA-2/86 Compiler	89	63
MODULA-2/86 with 8087 Support	129	103
MODULA-2/86 PLUS	189	147
Library Sources	99	89
Make Utility ... New	29	27
ROM Package ... New	199	179
Run Time Debugger	69	59
Turbo to Modula Translator	49	45
Utilities Package	49	45
Window Package	49	45
REPERTOIRE *for MODULA-2/86 by PMI* ... New	89	79

microport products

System V/AT *by Microport Systems* ... New	440	395
RUNTIME SYSTEM *(Operating Sys)* ... New	160	145
SOFTWARE DEVELOPMENT SYSTEM New	170	155
TEXT PREPARATION SYSTEM ... New	170	155
USER UPGRADE *3 to 8 Users* ... New	100	85

microsoft products

Microsoft BASIC Interpreter *for XENIX*	350	239
Microsoft C *with CodeView*	450	288
Microsoft COBOL Compiler	700	445
for XENIX	995	639
Microsoft COBOL Tools *with Debugger*	350	208
for XENIX	450	319
Microsoft FORTRAN Compiler	350	208
for XENIX	695	445
Microsoft Learning DOS ... New	50	CALL
Microsoft LISP *Common LISP*	250	175
Microsoft MACH 10 *Bundled package* ... New	549	CALL
Microsoft MACH 10 *Board* ... New	399	CALL
Microsoft Macro Assembler	150	98
Microsoft Mouse *Bus Version* ... *New Version*	175	139
Microsoft Mouse *Serial Version* ... *New Version*	195	149
Microsoft muMath *Includes muSIMP*	300	189
Microsoft Pascal Compiler	300	189
for XENIX	695	445
Microsoft QuickBASIC	99	75
Microsoft Sort	195	139
Microsoft Windows	99	68
Microsoft Windows Development Kit	500	329

other languages

CCS MUMPS *Single-User by MGlobal*	60	55
CCS MUMPS *Multi-User by MGlobal*	450	379
Janus/ADA C Pack *by R&R Software*	95	89
Janus/ADA D Pack *by R&R Software*	900	795
Methods *Smalltalk by Digitalk*	79	68
Personal REXX *by Mansfield Software*	125	109
Smalltalk/V *by Digitalk*	99	88
Smalltalk/Comm ... New	49	45
SNOBOL4+ *by Catspaw*	95	84

other products

Dan Bricklin's Demo Pgm *Software Garden*	75	59
FASTBACK *by 5th Generation Systems*	179	149
Informix *for DOS by RDS* ... New	795	639
Informix4GL *for DOS by RDS* ... New	995	799
InformixSQL *for DOS by RDS* ... New	795	639
Inside Track *from Micro Help* ... New	65	55
Instant Replay *by Nostradamus* ... New	90	79
Interactive EASYFLOW *by Haventree*	150	129
MACH 2 *by Micro Help* ... New	75	65
MKS Toolkit *with vi by MKS* ... New	139	119
OPT-Tech Sort *by Opt-Tech Data Proc*	149	115
Peeks 'n Pokes *from Micro Help* ... New	45	39
PrintQ *by Software Directions* ... New	89	84
Quilt Computing Combo Package	199	169
QMake *Program Rebuild Utility*	99	84
SRMS *Software Revision Mgmt Sys*	125	109
screenplay *all varieties by Flexus* ... New	CALL	CALL
Source Print *by Aldebaran Labs*	139	115
Taskview *by Sunny Hill Software*	80	65
VTEK, Term Emulator by Sci Endeavors ... New	150	129

phoenix products

Pasm86 *Macro Assembler* ... *New & Faster*	195	129
Pdisk *Hard Disk Utility* ... New	195	129
Pfantasy Pac	1295	879
Pfinish *Performance Analyzer*	395	239
Pfix-86 *Program Debugger*	195	135
Pfix-86 Plus *Symbolic Debugger*	395	239
PforCe *Comprehensive C Library*	395	239
Plink-86 Plus *Overlay Linker*	495	329
Pmaker *Make Utility* ... *New Version*	125	89
Pmate *Macro Text Editor*	195	119

Your Quality Connection ...

LOWEST PRICES

Since this ad is prepared in advance of publication, some of our current prices may be lower than what's advertised here. Call for latest pricing.

FREE SHIPPING

Orders within the USA are shipped FREE via standard UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges.

CREDIT CARDS

VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date.

CODs AND POs

CODs and Purchase Orders are accepted at no extra cost. POs with net 30-day terms are available to qualified US accounts.

FOREIGN ORDERS

Foreign and Canadian mail orders please include sufficient funds for shipping (excess payment will be refunded). Foreign orders (except Canada), please include $10 for customs form preparation. All transactions are in US dollars.

VOLUME ORDERS

Call for special pricing.

SOUND ADVICE

Our knowledgeable technical staff can compare products, answer technical questions and send you detailed product information tailored to your needs.

30-DAY GUARANTEE

Most of our products come with a 30-day return guarantee or a 30-day evaluation period. Please note that some products are restricted by their manufacturers from this guarantee. Call for more information.

CALL TOLL FREE

US 800-336-1166

CANADA 800-225-1166

OHIO 216-877-3781

CUSTOMER SERVICE 216-877-1110

Hours: Weekdays 8:30 AM to 8:00 PM EST.

Ohio customers please add 5% state sales tax.

Call or write for our FREE comprehensive price guide.

Prices are subject to change without notice.

Pre-C *Lint Utility* ... *New Version*	295	159
Ptel *Binary File Transfer Program*	195	119

polytron products

PolyBoost *The Software Accelerator* ... *New*	80	69
Polytron C Beautifier	49	45
Polytron C Library I	99	78
Polytron PowerCom *Communications*	179	139
PolyLibrarian *Library Manager*	99	78
PolyLibrarian II *Library Manager*	149	115
PolyMake *UNIX-like Make Facility*	99	78
PolyOverlay *Overlay Optimizer*	99	78
PolyWindows Products *All Varieties*	CALL	CALL
PolyXREF *Complete Cross Ref Utility*	219	179
PolyXREF *One language only*	129	109
PVCS *Version Control System*	395	325

softcraft products

Btrieve *ISAM Mgr with No Royalties*	250	194
Xtrieve *Query Utility* ... *New Version*	CALL	CALL
Rtrieve Report Option ... *New Version*	CALL	CALL
Btrieve/N *for Networks*	595	464
Xtrieve/N ... *New Version*	CALL	CALL
Rtrieve/N Report Option ... *New Version*	CALL	CALL

text editors

Brief *from Solution Systems*	195	CALL
Epsilon *Emacs-like editor by Lugaru*	195	159
KEDIT *by Mansfield Software*	125	105
PC/VI *by Custom Software Systems*	149	129
SPF/PC *by Command Technology Corp*	195	149
Vedit *by CompuView*	150	109
Vedit Plus *by CompuView*	185	139

turbo pascal utilities

See also Blaise, Borland and SoftCraft sections.

ALICE *Interpreter by Software Channels*	95	68
Btrieve *ISAM File Mgr* ... *See SoftCraft*	250	194
FirsTime for Turbo *by Spruce Tech*	75	59
Flash-up Windows *by Software Bottling*	75	68
HELP/Control *by MDS* ... *New*	125	109
On-line Help *from Opt-Tech Data Proc*	149	109
Screen Sculptor *by Software Bottling*	125	94
TDebugPLUS *by TurboPower Software* ... *New*	60	53
Turbo EXTENDER *by TurboPower Software*	85	68
Turbo Professional *by Sunny Hill*	70	49
TurboHALO *from IMSI* ... *New*	99	85
TurboPower Utilities *by TurboPower Sftwr*	95	84
TurboRef —by Gracon Services	50	45
TurboWINDOW *by MetaGraphics*	80	65

wendin products

Operating System Toolbox	99	84
PCNX *Operating system*	99	84
PCVMS *Similar to VAX/VMS*	99	79
XTC *Text editor with Pascal source*	99	79

xenix system v

See also Microport System V/AT section.

XENIX System V Complete System *by SCO*	1295	999
XENIX Development System	595	499
XENIX Operating Sys *Specify XT/AT*	595	499
XENIX Text Processing Package	195	149

xenix products

APL*PLUS/UNX *For AT XENIX by STSC*	995	695
Btrieve *ISAM File Mgr by SoftCraft*	595	464
C-ISAM *by Relational Database Sys* ... *New*	319	285
c-tree *ISAM Mgr w/Source by FairCom*	395	329
dBx *dBase/C Translator by Desktop AI*	550	499
dbVISTA *Single User w/Source by Raima*	495	425
dbVISTA *Multi User by Raima*	495	425
dbVISTA *Multi User w/Source by Raima*	990	845
Informix *by Relational Database Sys*	995	795
Informix4GL *by RDS*	1500	1199
InformixSQL *by RDS*	995	795
Lyrix *by SCO*	595	449
Micro Focus Level II Compact COBOL	1000	795
Forms-2	400	319
Level II ANIMATOR	600	479
Microsoft BASIC Interpreter	350	239
Microsoft COBOL Compiler	995	639
Microsoft COBOL Tools — with Debugger	450	319
Microsoft FORTRAN Compiler	695	445
Microsoft Pascal Compiler	695	445
Networks for XENIX *by SCO*	595	495
PANEL *Screen Designer byRyan-McFarland*	750	549
SCO Professional *Lotus clone by SCO*	795	595

Programmer's Connection Incorporated
136 Sunnyside Street
Hartville, Ohio 44632

programmer's connection

HIRE A PRO!

DDJ announces a new service: Programmers' Opportunities

Dr. Dobb's Journal of Software Tools has a 10-year history of serving professional programmers with the most useful technical information of any publication for the computer industry. Now, in our new Programmers' Opportunities section, we are offering our readers information to help them stay on top of the quickly changing technical-employment market.

We invite the top hardware, software, electronic, and instrument companies to list their computer-related career opportunities in *Dr. Dobb's Journal*. A limited number of companies will be given the chance to list jobs each month on a first-come, first-served basis.

For more information on how your company can hire a pro, write or call Gary George, 893 Monroe Dr., Atlanta, GA 30308; (404) 897-1923.

Our first responses are shown here as examples of what your listing will look like:

Position Title: Fortran Pgmr/Analyst
Position Location: Nashville TN
Company/Agency: Telco Research Corp.
Address: 1207 17th Avenue S.
City/State/Zip: Nashville TN 37075
Phone: 615-329-0031

Position Title: "C" Language Applications Pgmr
Position Location: Nashville TN
Company/Agency: Telco Research Corp.
Address: 1207 17th Avenue S.
City/State/Zip: Nashville TN 37075
Phone: 615-329-0031

Position Title: 6502 Assembly Pgmr (C64/128)
Position Location: Armonk NY
Company/Agency: Bodylog, Inc.
Address: 34 Maple Avenue
City/State/Zip: Armonk NY 10504
Phone: (914)-273-6480

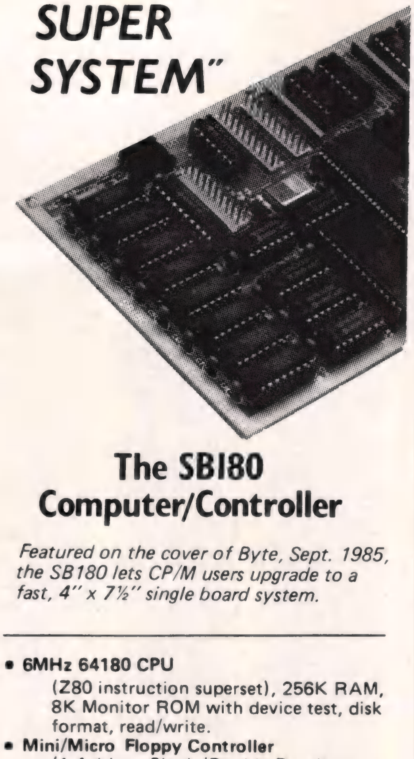
MANDELBROT PROGRAM

These Softstrips by Cauzin Systems contain the listings and object code for Howard Katz's Mandelbrot program. Strips 1 through 10 on this page and on page 71d contain the source code, in ASCII text format. Strips 1 and 2 on page 71d contain the actual application.

1 2 3 4 5

Byte Magazine called it,

"CIARCIA'S SUPER SYSTEM"



The SB180 Computer/Controller

Featured on the cover of Byte, Sept. 1985, the SB180 lets CP/M users upgrade to a fast, 4" x 7½" single board system.

- **6MHz 64180 CPU**
  (Z80 instruction superset), 256K RAM, 8K Monitor ROM with device test, disk format, read/write.
- **Mini/Micro Floppy Controller**
  (1-4 drives, Single/Double Density, 1-2 sided, 40/77/80 track 3½", 5¼" and 8" drives).
- **Measures 4" x 7½", with mounting holes**
- **One Centronics Printer Port**
- **Two RS232C Serial Ports**
  (75-19,200 baud with console port auto-baud rate select).
- **Power Supply Requirements**
  +5V +/-5% @500 mA
  +12V +/- 20% @40mA
- **ZCPR3 (CP/M 2.2/3 compatible)**
- **Multiple disk formats supported**
- **Menu-based system customization**

SB180-1
SB180 computer board w/256K bytes RAM and ROM monitor **$369.00**

SB180-1-20
same as above w/ZCPR3, ZRDOS and BIOS source............. **$499.00**

-Quantity discounts available-

NEW

COMM180-M-S

optional peripheral board adds 1200 bps modem and SCSI hard disk interface.

TO ORDER CALL TOLL FREE
1-800-635-3355

TELEX
643331

For technical assistance or to request a data sheet, call:
1-203-871-6170

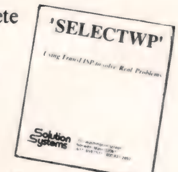
MICRO MINT

Micromint, Inc.
25 Terrace Drive
Vernon, CT 06066

Is LISP Right for Your Expert System?

Find Out — FREE

You can explore LISP by examining a complete sample problem. Call and we will send you a free source listing of "SELECTWP".* It prompts users for criteria and helps them choose which micro word processor to buy.



Look over the TransLISP syntax (COMMON LISP compatible). Your application will probably have similar characteristics.

Power & Flexibility

Do you get flexibility in PASCAL and C? Of course, but examine the listing of SELECTWP to see how much more power and flexibility you get. The LISP advantages:

- forward references make program flow fit the problem
- manipulate data structures of varying sizes
- create your own language to fit the problem domain
- avoid mundane, busy work required with traditional procedural languages
- powerful function and macro building facilities provide better data abstraction

Full refund if not satisfied in 30 days.

*SELECTWP includes 512 lines of LISP code and 335 lines of comments.

TransLISP gives You the Advantage

Using TransLISP for your expert system has several advantages over other AI tools. And you will see SELECTWP illustrate:

* the ability to control how decisions are made
* the freedom to assign weights and react to user choices
* the complete control you have over how a problem is solved, and interaction with the user

Nothing to lose

Examine LISP carefully by studying a practical program *free*.

Or buy TransLISP risk free. SELECTWP is just 1 of over 20 sample programs in the complete TransLISP system. The other sample programs include: an adventure game, a program to read dBASE SDF files, "Job Counselor" and more. Use the modular tutorial, the complete 300+ function LISP interpreter, and the online help, to get started in LISP in only a few hours.

Develop programs of up to 12000 lines on a 640K system or use TransLISP on a floppy only, 256K RAM machine. MSDOS.

Call 800-821-2492 for SELECTWP FREE. Or order the complete TransLISP system risk free for only $95.

Solution Systems

335 Washington St., Norwell, MA 02061 (617) 659-1571

Circle **no. 148** on reader service card.

The C Programmer's Assistant

C TOOLSET™

UNIX-like Utilities for Managing C Source Code

No C programmer should be without their assistant — C ToolSet. All of the utility programs are tailored to support the C language, but you can modify them to work with other languages too.

Source code in standard K&R C is included; and you are welcome to use it with any compiler (UNIX compatible) and operating system you choose.

12 Time Savers

DIFF - Compares text files on a line-by-line basis; use *CMP* for byte-by-byte. Indispensable for showing changes among versions of a program under development.

GREP - Regular expression search. Ideal for finding a procedural call or a variable definition amid a large number of header and source files.

FCHART - Traces the flow of control between the large modules of a program.

PP (C Beautifier) - Formats C program files so they are easier to read.

CUTIL - A general purpose file filter.

CCREF - Cross references variables used within a program.

CBC (curly brace checker) - checks for pairing of curly braces, parens, quotes, and comments.

Other utilities include *DOCMAKE*, *ASCII*, *NOCOM*, and *PRNT*.

Source code to every program is included!

Thorough User Support Text and Online

C ToolSet documentation contains descriptions of each program, a listing of program options (if any), and a sample run of the program.

On-line help gives you information on the programs and how to run them. Most of the programs respond to -? on the command line with a list of options.

Call 800-821-2492 to order C ToolSet risk-free for only $95.

Solution Systems™

335-D Washington St.,
Norwell, MA 02061
(617) 659-1571

Full refund if not satisfied in 30 days.

Requires MSDOS and 12K RAM

Circle **no. 152** on reader service card.

MANDELBROT PROGRAM

6 7 8 9 10 1 2

Softstrip

Softstrip

MANDELBROT PROGRAM

Listing One (Text begins on page 42.)

```
Listing One

; < MandelZoom.ASM >         Sat 30 Nov '85  h. katz
;                            Sun  4 May '86


INCLUDE MacTraps.D

String_Format      3                     ; pre-length DC.B and PEA Strings

XREF      Convert_2_Fixed_Point          ; Procedure defined in < Str2FP.ASM >

MACRO     Fix_Squared          Rn        =         ; a Mac-style macro
          clr.l      -(sp)                         ; ( Mac-Mac ? )
          move.l     {Rn}, -(sp)
          move.l     {Rn}, -(sp)
          _FixMul
          move.l     (sp)+, {Rn} |

MouseDown EQU          1        ; for _GetNextEvent
numToString       EQU    0         ; for _Pack7 conversions
stringToNum       EQU    1
Gray              EQU  -24         ; offset from QDVars Ptr
White             EQU   -8
portRect          EQU   16         ; offset from start of Window Record
pnPat             EQU   58         ; offset from start of Window Record

X_Screen_Offset   equu    4
Y_Screen_Offset   equ     4
Row_Pixels        equ   256
Col_Pixels        equ   256
PenSize           equ     2

HiLite_Off        equ     0
HiLite_On equ       1

Radio_Item_1      equ     9
Radio_Item_2      equ    10
Radio_Item_3      equ    11

X_Org_Item        equ    12        ; Item Numbers in Params DITL
Y_Org_Item        equ    13
Side_Length_Item  equ    14
Count_Item_1      equ    15

Org_Spacing       equ    24        ; Space tween X, Y, and S
Max_Count_Digits  equ     4        ; Num Digits in 'Count' Item Strings

Count_Str_X       equ     5        ; X_coord of Counts
Count_Str_Y       equ   114        ; Y_coord of 1st ( Max ) Count
Count_Str_Size    equ    10        ; Bytes wide

Legend_Plot_Item   equ       1
Legend_Quit_Item   equ       2

Pattern_Spacing    equ       30       ; Delta-Y for both Counts & Patts
Pattern_X equ       62      ; Left for Patts in Legend DLOG
Pattern_Y equ       86      ; Top for 1st Patt in Legend DLOG
Pattern_Size       equ        8       ; Bytes

X_Org_Scr_X        equ       10
X_Org_Scr_Y        equ       24

Time_Scr_X         equ       10
Time_Scr_Y         equ       16


          st        First_Entry(A5)
          sf        Radio_1_State(A5)
          st        Radio_2_State(A5)              ; default Pen is 2 X 2
          sf        Radio_3_State(A5)

          BSR       InitManagers
          BSR       Save_Mouse_State
          BSR       Draw_Menu_Title

MainLine

          BSR       Open_Params_DLOG

          tst.b     First_Entry(A5)
          BNE.s     @Set_Radios
          BSR       Reload_DITL                    ; 2nd time around -
                                                   ; get old Parameters
@Set_Radios

          sf        First_Entry(A5)

          BSR       Set_Radio_Buttons
          BSR       Get_Param_Items     ; Get User Choice / if OK, Toggle Radio
                                        ; Buttons, Convert & Save Counts
          bMI       Exit_To_Shell
          BSR       Save_Param_Items ; Save Str Counts / Convert 3 Fix-Pt Nums
          pea       paramsDLOGStorage
          _CloseDialog

          BSR       Draw_Mandel_Window
          BSR       Open_Legend_DLOG
          BSR       Draw_Patterns
          BSR       Draw_Org_Strings
```

(continued on page 74)

dBASE to the max.

Introducing a turbocharger for dBASE®.

Now the world's leading dBASE gurus, Jeb Long, Robert Byers and Wayne Ratliff, have combined their own personal utilities into a collection of dBASE programming tools that'll allow you to maximize your output while minimizing your time and effort.

Included are "secrets of the stars" on dBASE database repair and recovery, program structuring and cross referencing for dBASE programs, restricting access to a dBASE database, communications port control from within dBASE, menubar creation, saving and restoring screens, and much more.

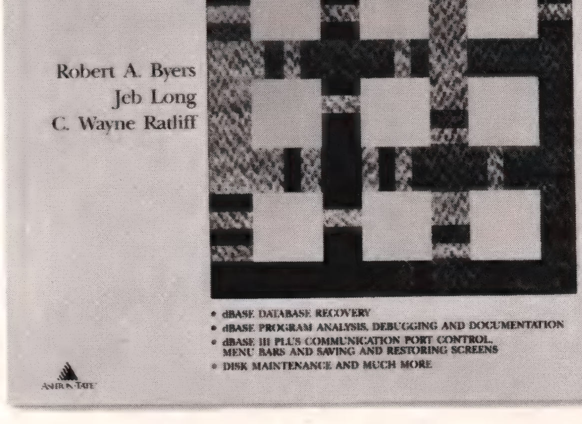
It's "insider information" you can't go to jail for using.

For merely $89.95 you have at your fingertips what the database honchos do: over a dozen DOS level programs to assist you with disk and directory management, time displays, and rapid searching for file names and switching between screen buffers.

Not to mention plenty of other neat stuff.

To order, or for more information or the name of your nearest dealer, call the Ashton-Tate® Publishing Group at (800) 437-4329, Ext. 221.

And put your dBASE into high gear.



Circle **no. 248** on reader service card.

Listing One (Listing continued, text begins on page 42.)

```
         BSR        Timer_On

         BSR        Do_Mandelbrot

         bMI        Exit_To_Shell        ; These 2 in case we've interrupted
         bHI        Do_Another           ; plotting in the middle

         BSR        Timer_Off
         BSR        Write_Time

Wait_4_Command

         BSR        Get_Next_Event
         bEQ        Wait_4_Command       ; No Event
         BSR        Was_Dialog_Event
         bEQ        Wait_4_Command       ; 0 => Hang Around a Bit
         bMI        Exit_To_Shell        ; - => Quit
                                         ; + => do another
Do_Another

         pea        MandelWindStorage
         _CloseWindow
         pea        LegendDLOGStorage
         _CloseDialog

         BRA        MainLine


Exit_To_Shell

         _ExitToShell


Save_Mouse_State   ; if the Mouse is Down on Launch, we'll
                   ; _SetPat and _Line for EVERY Point

         sf         Mouse_Down(A5)
         clr        -(sp)
         _Button
         tst        (sp)+
         beq        @rts
         st         Mouse_Down(A5)
@rts     RTS


Draw_Menu_Title

         move.l     #$000F0010, -(sp)
         _MoveTo
         pea        MBarTitle
         _DrawString
         RTS

Reload_DITL

         lea        TempStr, A2
         move       #0, D3

         move       #X_Org_Item, D4
         BSR        Get_Item_Text
         move.l     ItemHandle, -(sp)
         pea        X_Org_Str
         _SetIText
         move       #Y_Org_Item, D4
         BSR        Get_Item_Text
         move.l     ItemHandle, -(sp)
         pea        Y_Org_Str
         _SetIText
         move       #Side_Length_Item, D4
         BSR        Get_Item_Text
         move.l     ItemHandle, -(sp)
         pea        Side_Length
         _SetIText

         lea        TempStr, A2
         lea        Count_Strings, A3
         move       #0, D3
         move       #Count_Item_1, D4

@Reset_Counts

         BSR        Get_Item_Text
         move.l     ItemHandle, -(sp)
         move.l     A3, -(sp) ; Addr of Current Count_String
         _SetIText

         add.l      #Count_Str_Size, A3
         add        #1, D4               ; next Item Number in DLOG
         add        #1, D3               ; increment loop counter
         cmp        #4, D3               ; done all 4 ?
         BMI        @Reset_Counts        ; no

         RTS


Get_Next_Event

         clr        -(sp)
         move       #-1, -(sp)
         pea        EventRecord
         _GetNextEvent
         tst.b      (sp)+
         RTS


Was_Dialog_Event

         clr.b      -(sp)
         pea        EventRecord
         _IsDialogEvent
         tst.b      (sp)+                ; EQ = No Event
         bNE.s      @1                   ; NE = Was DLOG Event
         RTS

@1       clr.b      -(sp)
         pea        EventRecord
         pea        theDialog
         pea        ItemHit
         _DialogSelect
         tst.b      (sp)+
         bNE.s      Get_Legend_DLOG_Item

         clr.b      -(sp)
         pea        EventRecord
         pea        ParamsDLOGStorage
         pea        ItemHit
         _DialogSelect
         tst.b      (sp)+
         move       #0, D0
         RTS


Get_Legend_DLOG_Item

         move       ItemHit, D0
         cmp        #Legend_Plot_Item, D0
         bEQ        @Return_Plus
         cmp        #Legend_Quit_Item, D0
         bEQ        @Return_Minus
         move       #0, D0
         RTS


@Return_Minus                  ; = Quitting

         move       #-1, D0
         RTS

@Return_Plus                   ; = Do Another Mandelbrot

         move       #1, D0
         RTS


Timer_On

         clr.l      -(sp)
         _TickCount
         move.l     (sp)+, Start_Time(A5)
         RTS


Timer_Off

         _PenNormal
         move       #4, -(sp) ; Wake the Poor User
         _SysBeep

         clr.l      -(sp)
         _TickCount
         move.l     (sp)+, D3
         sub.l      Start_Time(A5), D3 ; ( Stop - Start ) in Ticks
         divu       #60, D3                    ; Num_Seconds (in Low Word)
         RTS
MenuRect dc         0, 10, 19, 200

Write_Time

         pea        MandelWindStorage
         pea        TempSTR
         _GetWTitle

         lea        TempSTR, a2
         move.l     a2, a3                     ; save start addr
         clr.l      d5                         ; clear out old junk
         move.b     (a2)+, d5          ; Length Byte
         adda.l     d5, a2                     ; point past last Char in Str

         lea        ' : ', a0                  ; addr of length byte
         clr.l      d1
         move.b     (a0)+, d1          ; save new length
         add.b      d1, d5
         move.b     d5, (a3)           ; put back new length byte
         sub        #1, d1

@Loop_1
         move.b     (a0)+, (a2)+               ; add new string to end
         dbra       d1, @Loop_1

         move.b     -(a2), d4          ; save last char

         ext.l      D3                         ; Elapsed Time in seconds
         move.l     D3, D0
         move.l     a2, a0
         move       #NumToString, -(sp)
         _Pack7

         clr.l      d1
         move.b     (a2), d1           ; save New Length Byte
         move.b     d4, (a2)           ; restore last Char of 1st String

         add.b      d1, d5                     ; new length
         move.b     d5, (a3)           ; and put back in Length Byte
         adda.l     d1, a2                     ; point to end of string
         adda.l     #1, a2                     ; points 1 past end

         lea        ' seconds', a1
         move.b     (a1)+, d1          ; save new Length Byte
         ext.w      d1
         add.b      d1, d5                     ; new total Length of Strings
         move.b     d5, (a3)           ; put it back in Length Byte
@Loop_2
         move.b     (a1)+, (a2)+               ; append 'Seconds' to end
         sub        #1, d1
         bhi        @Loop_2
```

(continued on page 76)

Now You Know Why BRIEF™ is BEST

"BRIEF is simple to learn and use and extremely sophisticated."

PC Magazine, July 1986

The Program Editor with the BEST Features

Since its introduction, BRIEF has been sweeping programmers off their feet. Why? Because BRIEF offers the features **MOST ASKED FOR** by professional programmers. In fact, BRIEF has just about every feature you've ever seen or imagined, including the ability to configure windows, keyboard assignments, and commands to **YOUR** preference. One reviewer (David Irwin, DATA BASED ADVISOR) put it most aptly, ''(BRIEF). . .is quite simply the best code editor I have seen.''

REGULAR EXPRESSION SEARCH

Regular expression searching is one of BRIEF's most powerful features. A regular expression is a series of "wildcards" that match pieces of your text. BRIEF supports a full set of regular expression characters similar to those found in UNIX including: beginning an end of line, groups, and the "closure" and "or" operators.

As Steve McMahon explained in Byte, "Not only does BRIEF make use of this marvelously general regular expression notation in its search facility, but its pattern recognition extends to its replacement (or translation) facility." "The usefulness of this facility for programmers who deal constantly with the regular expressions of formal languages is obvious . . ."

Every Feature You Can Imagine

Compare these features with your editor (or any other for that matter).

- FAST
- Full UNDO (N Times)
- Edit Multiple Large Files
- Compiler-specific support, like auto indent, syntax check, compile within BRIEF, and template editing
- Exit to DOS inside BRIEF
- Uses all Available Memory
- Tutorial
- Repeat Keystroke Sequences
- 15 Minute Learning Time
- Windows (Tiled and Pop-up)
- Unlimited File Size –(even 2 Meg!)
- Reconfigurable Keyboard
- Context Sensitive Help
- Search for ''regular expressions''
- Mnemonic Key Assignments
- Horizontal Scrolling
- Comprehensive Error Recovery
- A Complete Compiled Programmable and Readable Macro Language
- EGA and Large Display Support
- Adjustable line length up to 512

Program Editing YOUR Way

A typical program editor requires you to adjust your style of programming to its particular requirements – NOT SO WITH BRIEF. You can easily customize BRIEF to your way of doing things, making it a natural extension of your mind. For example, you can create ANY command and assign it to ANY key – even basic function keys such as cursor-control keys or the return key.

The Experts Agree

Reviewers at BYTE, INFOWORLD, DATA BASED ADVISOR, and DR. DOBB'S JOURNAL all came to the same conclusion – BRIEF IS BEST!

Further, of 20 top industry experts who were given BRIEF to test, 15 were so impressed they scrapped their existing editors!

NOT COPY PROTECTED

Solution Systems™

MONEY-BACK GUARANTEE
Try BRIEF ($195) for 30 days – If not satisfied get a full refund.
TO ORDER CALL (800-821-2492)

SOLUTION SYSTEMS, 335-D WASHINGTON ST., NORWELL, MA 02061, 617-659-1571

BRIEF is a trademark of UnderWare

Circle Reader Inquiry Number 142

Listing One (Listing continued, text begins on page 42.)

```
        pea     MandelWindStorage
        pea     TempSTR
        _SetWTitle

        RTS


Do_Mandelbrot

        pea     MandelWindStorage
        move.l  (sp), -(sp)                 ; copy WPtr for _SetWTitle trap
        _SetPort                     ; draw in this Window

        tst.b   Radio_1_State(A5)
        beq.s   @1
        pea     '1 X 1'
        bra     @SetTitle

@1      tst.b   Radio_2_State(A5)
        beq.s   @2
        pea     '2 X 2'
        bra.s   @SetTitle

@2      pea     '4 X 4'

@SetTitle
        _SetWTitle

@Set_Pen_Size
        move    #PenSize, D3
        tst.b   Radio_2_State(A5)   ; Draw with 2 X 2 Pen ?
        BNE     @Set_Pen            ; yes

        add     D3, D3                      ; Pen = 4 X 4
        tst.b   Radio_3_State(A5)   ; Draw with 4 X 4 ?
        BNE     @Set_Pen            ; yes
        move    #1, D3                      ; Draw with 1 X 1
@Set_Pen
        move    D3, Pix_Per_Pt(A5)
        move    D3, -(sp)
        move    (sp), -(sp)
        _PenSize

@Set_Plot_Size

        lea     MandelWindStorage, a0
        move    portRect+4(a0), d4 ; Window.Bottom
        move    #Y_Screen_Offset, d0
        sub     d0, d4                      ; frame at Bott
        sub     d3, d4                      ; move up 1 PenSize from Bott
        move    d4, Y_Start(a5)
        sub     d0, d4                      ; adjust for frame at Top
        move    d4, Num_Rows(a5)

        move    portRect+6(a0), d4 ; Window.Right
        move    #X_Screen_Offset, d0
        asl     #1, d0                      ; frame at Left & Right
        sub     d0, d4
        sub     d3, d4                      ; allow for penWidth
        move    d4, Num_Cols(a5)

@Get_C_Increment

        move.l  Y_side(A5), D0
        move    Num_Rows(A5), D5
        ext.l   d5
        divu    Pix_Per_Pt(A5), D5 ; = # of Plottable Pts on Y-Axis
        BSR     Get_Del_Factor              ; Del_Y returned as Fixed Pt
        move.l  D4, Del_C_imag(A5) ; in D4

        clr.l   -(sp)
        move    Num_Cols(a5), -(sp) ; numerator
        move    Num_Rows(a5), -(sp) ; denominator
        _FixRatio                   ; Fixed-Pt Ratio on stack
        move.l  (sp)+, d0           ; temp save it
        clr.l   -(sp)
        move.l  d0, -(sp)           ; Num_Cols/Num_Rows
        move.l  Y_Side(a5), -(sp)           ; x Y_Side
        _FixMul                             ; --------
        move.l  (sp)+, X_Side(a5)           ; = X_Side

        move.l  X_Side(A5), D0
        move    Num_Cols(A5), D5
        ext.l   d5
        divu    Pix_Per_Pt(A5), D5 ; = # of Plottable Pts on X-Axis
        BSR     Get_Del_Factor
        move.l  D4, Del_C_Real(A5)

        BRA     Continue


Get_Del_Factor

        move.l  D0, D3              ; save the fractional part
        swap    D0                  ; and get the whole part
        clr.l   -(sp)
        move    D0, -(sp) ; side ( integer part )
```

(continued on page 78)

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

DIGITAL RESEARCH COMPUTERS (214) 225-2309

CAF 8 MHZ AT COMPATIBLE - $1399

* PRODUCED BY CAF: A $500 MILLION/YEAR TAIWANESE CONGLOMERATE.
* BURNED IN FOR 48 HRS. * INCLUDES DOS 3.1 AND MANUALS.
* 512K RAM, EXPANDABLE TO 1MEG ON M.B. * 1.2 MEG FLOPPY INCLUDED.
* 80286 WITH OPTIONAL 80287 * ON BOARD REAL TIME CLOCK.
* AT STYLE KEYBOARD WITH ENLARGED ENTER AND SHIFT.
* 195W POWER SUPPLY * COMPLETE SERVICE MANUAL AVAILABLE: $40.
* COMPLETE SYSTEM DOCUMENTATION INCLUDED, PLUS DOS MANUALS.
* FLOPPY AND HARD DISK CONTROLLER CARD INCLUDED.
* ERSO BIOS FOR MAXIMUM AT SOFTWARE COMPATIBILITY.
* LIMITED 1-YEAR FACTORY WARRANTY. * FCC CERTIFIED!
* 1.2 MEG FLOPPY CAN READ AND WRITE 360K DISKETTES. ADD $15 UPS.

CMI 6640 33 MB. HARD DISK 39 MS - AT COMPATIBLE. VERY LIMITED STOCK — $325

MEX-PC MODEM SOFTWARE PACKAGE $69.95

A FANTASTIC COMMUNICATIONS PACKAGE WITH FEATURES TOO NUMEROUS TO LIST. SEE MAY/JUNE '86 MICRO SYSTEMS JOURNAL FOR FULL REVIEW OR CALL FOR BROCHURE. SUPPORTS COLOR, KERMIT PROTOCOL, Y MODEM BATCH, VT100 EMULATION, REMOTE OPERATION, ON LINE HELP, ETC. RUNS UNDER PC/MS DOS. 180 PAGE MANUAL & BINDER.

MEX-PACK LIST IS $99.95 — **SPECIAL $69.95**

STANFORD TURBO XT - $399

4.77 OR 8 MHZ * FCC CERTIFIED * 640K MOTHER BOARD (O K) * .135 WATT P.S. * XT-AT 5160 KEYBOARD * FLIP TOP CASE * 8 EXPANSION SLOTS * ASSEMBLED AND TESTED * WITH BIOS * FLOPPY CONTROLLER CARD & CABLES * WITH DOCUMENTATION. ADD $12 UPS.

A.R.C. XT TURBO - $529

4.77 OR 7.4 MHZ * 8 EXPANSION SLOTS * SLIDE OUT PC CASE * 5160 XT-AT KEYBOARD * MS DOS 3.1 + BASIC DISKETTE * 256K RAM, EXPANDABLE TO 640K * 4 LAYER JAPANESE MOTHER BOARD * FULLY TESTED AND BURNED IN. * SUPERIOR, CLEAN BIOS. * 135 WATT P.S. * SUPERB DOCUMENTATION * FCC CERTIFIED. ADD $15 UPS.

XT OPTIONS:

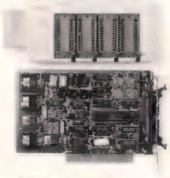
* V20.8 MHZ NEC PROCESSOR. REPLACES 8088. - $12.95
* TWO 1/2 HT. D.S. DD. FLOPPIES INSTALLED: $175/Pair
* 256K DRAM. 150 NS. INSTALLED AND TESTED: $28/9
* 64K DRAM. 150 NS. INSTALLED AND TESTED: $10/9
* HARD DISK: ST225 WITH CONTROLLER CABLES & FORMATTED: $425
* ZUCKER COLOR CARD WITH PARALLEL PRINTER PORT: $86
* MONOCHROME/GRAPHICS VIDEO CARD WITH PRINTER PORT: $89
* PLEASE CALL FOR MONITOR PRICING

S100 BUSS LIVES! WE STILL SELL S100 PRODUCTS AND CP/M SINGLE BOARDS. FREE CATALOG.

SCHEMA: Schematic Capture and Drawing Package

WORKS WITH PC'S AND COMPATIBLES. EASY TO USE WITH OFF THE SHELF PRINTERS, MOUSES, VIDEO CARDS, ETC. COMES COMPLETE WITH A HUGE COMPONENT DATABASE. SUPPORTS MONOCHROME, COLOR, OR EVEN EGA. FREE BROCHURE AND DEMO DISKETTE ON REQUEST. LIST $500 – **SPECIAL $396.15**

NEW! PC/XT EPROM PROGRAMMER $199

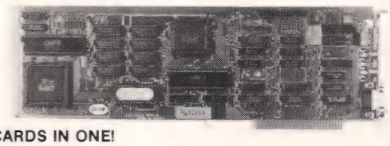


* LATEST DESIGN * PROGRAMS UP TO 4 DEVICES AT ONE TIME * FEATURES EASY TO USE MENU DRIVEN SOFTWARE THAT RUNS UNDER PC OR MS-DOS. * USES AN INTELLIGENT PROGRAMMING ALGORITHM FOR SUPER FAST (8X) EPROM BURNING. * THIS PLUG-IN BOARD ATTACHES TO AN EXTERNAL MINI CHASSIS CONTAINING 4 TEXTOOL Z.I.F. SOCKETS. * NO PERSONALITY MODULES REQUIRED * AUTOMATIC VPP SELECTION: 12.5V, 21V, OR 25V. * EPROM DATA CAN ALSO BE LOADED FROM OR SAVED TO A DISKETTE. * PROGRAMMING SOFTWARE SUPPORTS: 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256, 27256A, 27512, AND 27512A. * ASSEMBLED AND TESTED, BURNED. IN WITH MANUAL. $199 WITH SOFTWARE.

MICROTEK 2 M.B. EXPANDED MEMORY BOARD $195 (O. K)

MEETS LOTUS/INTEL/MICROSOFT SPEC. FOR EXPANDED MEMORY. FOR USE AS EXPANDED MEMORY (EMS), ELECTRONIC DISK, OR PRINT SPOOLER. WITH A SUPERB MANUAL AND INSTALLATION SOFTWARE. FOR PC/XT OR MOST COMPATIBLES. FOR 2 MEGABYTES OF 150 NS RAM (72 DEVICES) INSTALLED AND BURNED IN ADD $225.

PC/XT AT ENHANCED GRAPHICS ADAPTER

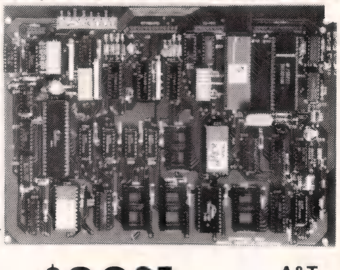


$279

* 3 CARDS IN ONE!
* 100% IBM COMPATIBLE.
* COMPATIBLE WITH IBM EGA, COLOR GRAPHICS ADAPTER, OR MONOCHROME ADAPTER.
* 256K VIDEO RAM! (4 TIMES MORE THAN IBM!)
* DUAL FREQUENCY OUTPUT FOR EITHER EGA OR STANDARD RGB COLOR MONITORS.
* PERFECT MATE FOR NEC MULTISYNC COLOR MONITOR!
* LIGHT PEN INPUT.
* FULL 16 COLORS.
* MFG. BY D.F.I., THE PREMIER FAR-EAST ADD-ON BOARD MAKER.
* A SUPERIOR BOARD AT ABOUT 1/3 THE COST OF IBM EGA!
* ASSEMBLED, TESTED, AND BURNED IN. WITH MANUAL.
* USES CHIPS & TECH. VLSI 4 CHIP SET.
* FIELD PROVEN BIOS.

ZRT-80 CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.



FEATURES:

* Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
* RS232 at 16 BAUD Rates from 75 to 19,200.
* 24 x 80 standard format (60 Hz).
* Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
* Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
* Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
* 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
* Composite or Split Video.
* Any polarity of video or sync.
* Inverse Video Capability.
* Small Size: 6.5 x 9 inches.
* Upper & lower case with descenders.
* 7 x 9 Character Matrix.
* Requires Par. ASCII keyboard.

FOR 8 IN. SOURCE DISK (CP/M COMPATIBLE) ADD $10

$89.95 #ZRT-80 A&T ADD $50

(COMPLETE KIT, 2K VIDEO RAM)

BLANK PCB WITH 2716 CHAR. ROM. 2732 MON. ROM

$49.95

SOURCE DISKETTE - ADD $10

SET OF 2 CRYSTALS - ADD $7.50

THE NEW 65/9028 VT ANSI VIDEO TERMINAL BOARD!

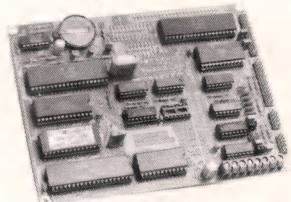
* FROM LINGER ENTERPRISES *

A second generation, low cost, high performance, mini sized, single board for making your own RS232 Video Terminal. Use as a computer console or with a MODEM for hook up to any of the telephone-line computer services.

FEATURES:

* Uses the new SMC 9028 Video Controller Chip coupled with a 6502A CPU.
* RS-232 at 16 Baud Rates from 50 to 19,200
* On board printer port!
* 24 X 80 format (50/60 Hz).
* For 15,750 Hz (Horiz.) monitors.
* 3 Terminal Modes: H-19, ADM3A, and ANSI X 3.64-1979
* Wide and thin-line graphics.
* White characters on black background or reversed.
* Character Attributes: De-Inten, Inverse, Underline and Blank.
* Low Power: 5VDC @ .7A, ± 12VDC @ 20MA.
* Mini size: 6.5 X 5 inches.
* Composite or split video.
* 5 X 8 Dot Matrix characters (U/L case) with descenders.
* Answer back capability.
* Battery backed up status memory.
* For ASCII parallel keyboard.

MICRO SIZE!



$99.95 (Full Kit)

ADD $40 FOR A&T

SOURCE DISKETTE: PC/XT FORMAT 5¼ IN. $15

NEW OPTION

PC/XT keyboard ROM. Allows use of IBM style keyboard. $15

Digital Research Computers

P.O. BOX 381450 • DUNCANVILLE, TX 75138 • (214) 225-2309

TERMS: Add $3.00 postage. Orders under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50 add 85¢ for insurance. Prices subject to change without notice.

*TM OF DIGITAL RESEARCH INC. (CALIF.) *IBM TM OF INT. BUS. MACHINES. WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

Circle no. 87 on reader service card.

Listing One (Listing continued, text begins on page 42.)

```
        move      D5, -(sp) ; pts per side
        _FixRatio           ; = length ( integer ) / point
        move.l    (sp)+, D4 ; save the ( int ) fraction

        sf        D6                ; positive fraction
        tst       D3
        bpl       @1
        st        D6                ; negative fraction
        lsr       #1, D3            ; zero the hi bit so _FixRatio doesn't
                                    ; think the number is Negative
@1      clr.l     -(sp)
        move      D3, -(sp) ; side ( fract part )
        move      D5, -(sp) ; pts per side
        _FixRatio           ; = length ( fract ) / point
        move.l    (sp)+, D3

        swap      D3                ; move the 'integer' part of the'fraction'
        tst.b     D6                ; back into the fractional lo word
        bpl       @2
        lsl       #1, D3            ; restore the 'negative' hi bit

@2      and.l     #$FFFF, D3
        add.l     D3, D4

        RTS

Continue

        clr       Row_Count(A5)
        move.l    Y_Origin(A5), C_Imag(A5)
        move      Y_Start(a5), Y_Current(A5)
Do_Next_Row

        clr       Col_Count(A5)

        move      #X_Screen_Offset, -(sp)    ; For next row
        move      Y_Current(A5), -(sp)       ; move absolute to start
        _MoveTo

        st        First_Pt(A5)               ; 1st_Point := TRUE;

        move.l    X_Origin(A5), C_Real(A5)   ; for start of new row

        BSR       Do_Points

        move      Row_Count(A5), D0
        add       Pix_Per_Pt(A5), D0
        move      D0, Row_Count(A5)
        cmp       Num_Rows(A5), D0

        bMI       @CheckDLOG
        move      #0, d0
        BRA       @Return_To_Mainline

@CheckDLOG

        BSR       Get_Next_Event
        bEQ.s     @Setup_Next_Row
        BSR       Was_Dialog_Event
        bEQ.s     @Setup_Next_Row

@Return_To_Mainline

        RTS

@Setup_Next_Row

        move      Pix_Per_Pt(A5), D0
        sub       D0, Y_Current(A5)

        move.l    C_imag(A5), D0             ; set up Y for next row
        add.l     Del_C_Imag(A5), D0
        move.l    D0, C_Imag(A5)

        BRA.s     Do_Next_Row


Do_Points

        move.l    C_Real(A5), D5     ; Initialize Z = C for new point
        move.l    C_Imag(A5), D6
        move      #1, Iter_Count(A5) ; Do up to Counts(A5) times per Point
        lea       Patterns, A4       ; reset Pattern Ptr
Iterate

        move.l    D5, D3                     ; Save Current Z_Real
        move.l    D6, D4                     ; Save Current Z_Imag

        Fix_Squared        D3                ; Z_Real^2
        Fix_Squared        D4                ; Z_Imag^2
        move.l    D4, D7
        add.l     D3, D7                     ; Size^2 = Z_Real^2 + Z_Imag^2

@Test_Size
        move      Iter_Count(A5), D0
        cmp.l     #$40000, D7                ; Size^2 > 4 means TIME TO PLOT
        BHI.s     @Plot

@Test_Count
        add       #1, D0
        move      D0, Iter_Count(A5)
        cmp       Counts(A5), D0
        BPL.s     @Plot

@Get_New_Z
        sub.l     D4, D3                     ; Z_Real = Z_Real^2 - Z_Imag^2

        clr.l     -(sp)
        move.l    D5, -(sp)
        move.l    D6, -(sp)
        _FixMul
        move.l    (sp)+, D6         ; Z_Real * Z_Imag
        add.l     D6, D6                     ; Z_Imag = 2 * Z_Real * Z_Imag
        move.l    D3, D5                     ; Z_Real
        add.l     C_Real(A5), D5
        add.l     C_Imag(A5), D6
        BRA.s     Iterate

@Plot
        BSR       Get_Pattern       ; A4 = Ptr to New Pattern

        tst.b     First_Pt(A5)      ; IF NOT 1st_Point
        BEQ       @Test_Mouse       ;   Test_Mouse ( see if batching )
        sf        First_Pt(A5)      ;     ; ELSE
                                    ;   1st_Point = FALSE;
        move.l    A4, A2            ;   Old_Pat := New_Pat;
        BRA       @Set_Pattern

@Test_Mouse

        tst.b     Mouse_Down(A5)    ; IF NOT Batch_Plot
        BNE.s     @Draw_Line        ;    Draw_Line ( Old_Pat )
        cmp.l     A2, A4            ; ELSE
        BNE       @Draw_Line        ;   IF NOT ( New_Pat = Old_Pat )
                                    ;     Draw_Line ( Old_Pat )
                                    ;   ELSE
                                    ;     Init_Line_Amount;
        add       Pix_Per_Pt(A5), A3;     Do_Next_Pt;
        BRA       @Skip_Draw

@Draw_Line

        move      A3, -(sp)
        move      #0, -(sp)
        _Line                       ; Draw_Line ( Old_Pat )

@Set_Pattern

        move.l    A4, -(sp) ; set the New Pattern
        _PenPat

        move.l    A4, A2            ; Old_Pat := New_Pat
        move      Pix_Per_Pt(A5), A3

@Skip_Draw

        move      Col_Count(A5), D0
        add       Pix_Per_Pt(A5), D0
        move      D0, Col_Count(A5)

        cmp       Num_Cols(A5), D0
        BMI.s     @Update_Z_Real

; we've finished the Line - if we need to draw to finish up
; do it here

        cmp       Pix_Per_Pt(A5), A3
        BEQ       @rts                        ; we've just drawn

        move      A3, -(sp)         ; else draw what we didn't
        move      #0, -(sp)
        _Line

@rts    RTS

@Update_Z_Real
        move.l    C_Real(A5), D0
        add.l     Del_C_Real(A5), D0
        move.l    D0, C_Real(A5)

        BRA       Do_Points

Get_Pattern       ; Point to a New PenPat, according to which
                  ; Range the Iter_Count ( D0 ) falls in

        cmp       0+Counts(A5), D0   ; >= Black
        BPL.s     @0

        add       #8, A4
        cmp       2+Counts(A5), D0   ; >= DarkGray
        BPL.s     @0
        add       #8, A4
        cmp       4+Counts(A5), D0   ; >= LtGray
        BPL.s     @0
        add       #8, A4
        cmp       6+Counts(A5), D0   ; >= White
        BPL.s     @0
        add       #8, A4             ; <  Gray
@0      RTS


Open_Params_DLOG

        clr.l     -(sp)             ; space for funct result
        move      #100, -(sp)
        pea       ParamsDLOGStorage
        move.l    #-1, -(sp)        ; in front of everything
        _GetNewDialog
        move.l    (sp)+, d0
        RTS

Set_Radio_Buttons

        move      #HiLite_On, D3

        tst.b     Radio_1_State(A5)
        BPL       @2
        move      #Radio_Item_1, D4
```

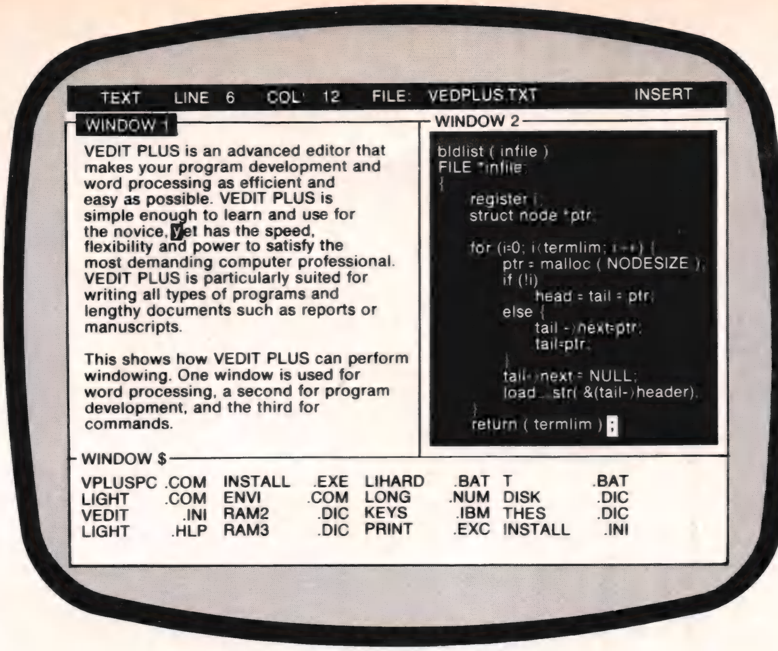
(continued on page 80)

MANDELBROT PROGRAM

Listing One (Listing continued, text begins on page 42.)

```
        BRA     HiLite_Control

@2      tst.b   Radio_2_State(A5)
        BPL     @3
        move    #Radio_Item_2, D4
        BRA     HiLite_Control

@3      move    #Radio_Item_3, D4
        BRA     HiLite_Control

HiLite_Control

        pea     ParamsDLOGStorage
        move    D4, -(sp) ; ItemNumber
        pea     ItemType
        pea     ItemHandle
        pea     ItemBox
        _GetDItem
        move.l  ItemHandle, -(sp)
        move    D3, -(sp)
        _SetCtlValue

        RTS

Get_Param_Items

        pea     ParamsDLOGStorage  ; Select 'X_Org' Parameter
        move    #X_Org_Item, -(sp) ; for Quick Replacement
        move    #0, -(sp)
        move    #32767, -(sp)
        _SelIText

ModalDLOG

        clr.l   -(sp)              ; no filterProc
        pea     ItemHit
        _ModalDialog

        move    ItemHit, D0
        tst     D0
        BEQ     ModalDLOG
        cmp     #1, D0             ; Clicked 'OK' = We're Done Dialoging ?
        BEQ     Validate_Items     ; yes - Validate & Convert numeric entries
        cmp     #2, D0             ; Clicked 'Quit' ?
        BEQ     Set_Exit_Flag      ; yes - tell MainLine

        cmp     #Radio_Item_1, D0 ; Clicked a Radio Button for penSize ?
        bMI     ModalDLOG ; no - wait for 'OK' or 'Quit'
        cmp     #Radio_Item_3, D0
        bHI     ModalDLOG          ; no - wait for 'OK' or 'Quit'
        BSR     Toggle_Radio_Buttons      ; yes
        BRA     ModalDLOG          ; and wait for 'OK' or 'Quit'

Set_Exit_Flag

        move    #-1, D0
        RTS

Toggle_Radio_Buttons

        move    D0, D5                     ; (D0 gets trashed by ROM calls)
        move    #HiLite_Off, D3            ;
        move    #Radio_Item_1, D4  ; turn off Everything
        BSR     HiLite_Control
        move    #Radio_Item_2, D4
        BSR     HiLite_Control
        move    #Radio_Item_3, D4
        BSR     HiLite_Control

        sf      Radio_1_State(A5)  ; Flag them as OFF
        sf      Radio_2_State(A5)
        sf      Radio_3_State(A5)

        move    #HiLite_On, D3             ; turn ON the Radio Item
        move    D5, D4                     ; that was Clicked
        BSR     HiLite_Control
        cmp     #Radio_Item_1, D5  ; and Flag the apt Item
        BNE     @2                         ; as ON
        st      Radio_1_State(A5)
        RTS

@2      cmp     #Radio_Item_2, D5
        BNE     @3
        st      Radio_2_State(A5)
        RTS

@3      st      Radio_3_State(A5)
        RTS
```

(continued on page 82)



VEDIT PLUS

MULTIPLE WINDOWS

POP-UP MENUS

KEYSTROKE MACROS

EXECUTE DOS PROGRAMS

For over six years VEDIT has been the choice of professionals who demand the most powerful editing software available. CompuView has once again enhanced this power with the latest VEDIT PLUS - you can now open windows to simultaneously edit several files, access editing functions with pop-up menus, use keystroke macros to speed editing and run other programs within VEDIT PLUS.

Whether your needs are program development, technical writing or word processing, VEDIT PLUS is your answer. VEDIT PLUS is simple enough to learn for the novice, yet has the speed, flexibility and power to satisfy the most demanding computer professional. Its powerful macro programming language helps you eliminate repetitive editing tasks.

If you take your editing seriously, you need VEDIT PLUS. With over 40,000 users, you can depend on VEDIT PLUS to perform consistently and reliably. As have GE, EDS, U.S. Navy, GM, Sperry and many others. VEDIT PLUS supports color windows on the IBM CGA & EGA and even windows on most CRT terminals. Available for MS-DOS, PCDOS, CP/M-86 and CP/M-80. List price $185.

> "To sum things up, VEDIT PLUS is a small, fast, sophisticated editor with a wealth of features and a good macro language. It offers many rewards for the dedicated programmer."
> **Computer Language, Chris Wolf, Scott Lewis, Mark Gayman 6/86**

> "VEDIT PLUS is a wholly remarkable program: blindingly fast, extremely powerful, and highly flexible."
> **Profiles Magazine, Robert Lavenda 4/86**

VEDIT PLUS FEATURES

- Simultaneously edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname and CP/M user number support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 text registers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size, any CRT.
- Optimized for IBM PC/XT/AT. Also 132 column & up to 70 lines.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I or PASCAL.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert WordStar and mainframe files.
- Print any portion of file; separate printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts keyboard input, 17 bit algebraic expressions, variables.
- CRT emulation within windows, Forms entry.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Main menu

Circle **no. 122** on reader service card.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

Listing One (Listing continued, text begins on page 42.)

```
Validate_Items

        move    #0, D3              ; Count - 1
        lea     Count_Strings, A2
        move    #Count_Item_1, D4   ; Item # of 1st Count (MaxCount)

@0      BSR     Get_Item_Text       ; get the next Item Text
        BSR     Convert_2_Int       ; Convert theString to Integer

        add     #10, A2             ; point to next String
        add     #1, D4              ; and its Item Number
        add     #1, D3              ; for Next Count Range
        cmp     #4, D3              ; Done all 4 Count Ranges ?
        BMI     @0                  ; not yet

        RTS                         ; Return to MainLine


Get_Item_Text       ; A2 points to the String

        pea     ParamsDLOGStorage   ; DLOG Ptr
        move    D4, -(sp)           ; Item Number
        pea     ItemType            ; Not Used
        pea     ItemHandle          ; passed to following ROM call
        pea     ItemBox             ; Not Used
        _GetDItem

        move.l  ItemHandle, -(sp)
        move.l  A2, -(sp)
        _GetIText

        RTS


Convert_2_Int

        move.l  A2, A0
        move    #StringToNum, -(sp)
        _Pack7                      ; Convert Count to Numeric
        move    D3, D5              ; Which Count Range ?
        add     D5, D5              ; Words => Bytes for Offset
        lea     Counts(A5), A0
        move    D0, 0(A0, D5)       ; Index & Save the Count
                                    ; ( Ignore the Hi Byte )
        RTS


Save_Param_Items

        move    #0, D3                          ; D3 not used here
        move    #X_Org_Item, D4
        lea     X_Org_Str, A2                   ; Following routine deposits
        BSR     Get_Item_Text                   ; DITL text in (A2)

        ; A2 (input) points to Decimal DITL String
        ; D0 (returned) contains Fixed-Point Conversion

        BSR     Convert_2_Fixed_Point           ; XREF routine to convert from
        move.l  D0, X_Origin(A5)    ; STR format to Fixed-Point
                                                ; format via SANE intermediary

        move    #Y_Org_Item, D4
        lea     Y_Org_Str, A2
        BSR     Get_Item_Text
        BSR     Convert_2_Fixed_Point
        move.l  D0, Y_Origin(A5)

        move    #Side_Length_Item, D4
        lea     Side_Length, A2
        BSR     Get_Item_Text
        BSR     Convert_2_Fixed_Point
        move.l  D0, Y_Side(A5)

        RTS


Draw_Org_Strings
        move    #X_Org_Scr_X, -(sp)
        move    #X_Org_Scr_Y, D3
        move    D3, -(sp)
        _MoveTo
        pea     'X  '
        _DrawString
        pea     X_Org_Str
        _DrawString

        move    #X_Org_Scr_X, -(sp)
        add     #Org_Spacing, D3
        move    D3, -(sp)
        _MoveTo
        pea     'Y  '
        _DrawString
        pea     Y_Org_Str
        _DrawString

        move    #X_Org_Scr_X, -(sp)
        add     #Org_Spacing, D3
        move    D3, -(sp)
        _MoveTo
        pea     'S  '
        _DrawString
        pea     Side_Length
        _DrawString

        RTS

Open_Legend_DLOG

        clr.l   -(sp)               ; space for Funct result
        move    #101, -(sp)
        pea     LegendDLOGStorage
        move.l  #-1, -(sp)          ; in front of everything
        _GetNewDialog
        move.l  (sp), -(sp)
        _SetPort            ; so Title prints
        _SelectWindow               ; so _DialogSelect works

        RTS


Draw_Patterns

        clr     -(sp)               ; Save Digit Char Width
        move    #'1', -(sp)         ; in D4 for Right-Justifying
        _CharWidth
        move    (sp)+, D4

        move    #Count_Str_Y, Legend_Y_Pos(A5)
        move    #0, D3
        lea     Count_Strings, A3   ; Addr of 1st Count Str
@Draw_Counts

        move    #Count_Str_X, -(sp)
        move    Legend_Y_Pos(A5), -(sp)
        _MoveTo

        cmp.b   #Max_Count_Digits, (A3)     ; Truncate STRs if too long
        BMI     @0
        move.b  #Max_Count_Digits, (A3)

@0      clr.l   D0                  ; Right-Justify Count_Strings
        move    #Max_Count_Digits, D0
        clr     D1
        move.b  (A3), D1  ; Byte Count for String
        sub     D1, D0              ; Del_Digits = Max_Digits - Actual Digits
        mulu    D4, D0              ; times Digit Char_Width
        move.l  D0, -(sp) ; = amount to space over
        _Move                       ; Relative Move

        move.l  A3, -(sp)
        _DrawString                 ; Write the Count Range Str

        move.l  (A5), A2            ; QD Vars Ptr
        pea     Gray(A2)
        _PenPat
        move.l  #$FFFB0006, -(sp)   ; Move Up & Over a Bit
        _Move                               ; Draw a Short Gray Line to
        move.l  #$0000000D, -(sp)   ;  separate the Patt_Rects
        _Line
        _PenNormal                          ; Back to Black for Next String

        move    Legend_Y_Pos(A5), D0        ; move down for Next String
        add     #Pattern_Spacing, D0
        move    D0, Legend_Y_Pos(A5)
        add.l   #Count_Str_Size, A3; point to Next String

        add     #1, D3
        cmp     #4, D3
        BMI     @Draw_Counts

        move    #Pattern_Y, Legend_Y_Pos(A5)
        move    #0, D3
        lea     Patterns, A3

@Draw_Patterns

        move    Legend_Y_Pos(A5), D0        ; Top
        swap    D0
        move    #Pattern_X, D0              ; Left

        lea     TempRect, A0
        move.l  D0, (A0)+           ; TopLeft
        add.l   #$00130013, D0                      ; 19 X 19
        move.l  D0, (A0)            ; BottomRight

        pea     TempRect
        move.l  (sp), -(sp)                 ; push 2 copies of Rect Addr
        move.l  (sp), -(sp)
        _FrameRect
        move.l  #$00010001, -(sp)
        _InsetRect
        move.l  A3, -(sp)
        _FillRect

        move    Legend_Y_Pos(A5), D0        ; move down for Next String
        add     #Pattern_Spacing, D0
        move    D0, Legend_Y_Pos(A5)
        add.l   #Pattern_Size, A3  ; point to Next Pattern

        add     #1, D3
        cmp     #5, D3
        BMI     @Draw_Patterns

        RTS


Draw_Mandel_Window

        clr.l   -(sp)
        move    #101, -(sp)
        pea     MandelWindStorage
        move.l  #-1, -(sp)
        _GetNewWindow
        _SetPort            ; nuthin hops if we don't do this

        lea     MandelWindStorage, A0
        pea     portRect(A0)
        _EraseRect

        RTS


InitManagers

        pea     -4(A5)
        _InitGraf
        _InitFonts
        _InitWindows
```

(continued on page 84)

ECOSOFT HOLIDAY SPECIALS

EVERYTHING FOR THE C PROGRAMMER

If you know someone (perhaps yourself!) that wants to learn the C programming language, have we got a deal for you. For a limited time, you get:

Eco-C88 C compiler	$59.95
CED Program editor	29.95
Developer's Library Source Code	25.00
ISAM File Handler	15.00
Flexi-Graph Graphics Package	39.95
Eco-Lib MSDOS Compatible Librarian	29.95
Ecosoft Windowing Library	29.95
C Programming Guide book (Que Corp)	20.00
C Self Study Guide book (Que Corp)	17.00
C Programmer's Library book (Que Corp)	22.00
Total:	$288.75
Holiday Special:	***$175.00***
Save Over:	$110.00!!!

The Holiday Special has everything you need for your C programming needs. Eco-C88 is a full K&R C compiler (less bit fields), with over 200 library functions, source code to our cc and mini-make utilities, and generates code rivaling compilers costing 10 times as much. You also get the source code to our library, an MSDOS-obj compatible librarian, one of the neatest graphics packages around, an easy-to-use windowing package plus a library of books to teach you the C language. An unbelievable package at an unbeatable price of only $175.00.

Microstat® Microstat Statistics Package

Since 1977, Microstat has been one of the most popular statistics packages available for microcomputers. Just some of its features include:

- Data Managment Subsystem
- Data transformations
- Convert External data (e.g. Lotus, Dbase 11. etc.)
- Descriptive Statistics
- Hypothesis Testing
- Crosstabs and Chi-Square
- ANOVA
- Multiple Regression (with Stepwise)
- Correlation Analysis
- Time Series Analysis
- Nonparametrics
- Probability Distributions
- Scatterplots
- Factorials, Permutations

Holiday Special: $250.00

plus many other unique and easy-to-use features. Microstat is available for all MSDOS machines or any CP/M80 system using a Z80 CPU.

The regular price is a bargain at $375.00. At the Holiday Special price of $250.00, it's a steal!

To take advantage of these special prices, we must have your order in our offices by Jan. 15, 1987.
To order, call or write:

1-800-952-0472 (for orders)
or
1-317-255-6476 (tech. info.)

ECOSOFT INC.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

AMERICAN EXPRESS MasterCard VISA

Trademarks: Ecosoft: Microstat, Eco-C88: Microsoft: MSDOS: New Orleans General Data Services: Flexi-Graph

Circle **no. 89** on reader service card.

Listing One

(Listing continued, text begins on page 42.)

```
        _InitMenus
        clr.l     -(sp)
        _InitDialogs
        _TEInit
        _InitCursor         ; the slings and arrows . . .
        RTS

; ---------------- Constants ( in Code Space ) -----------------

MBarTitle dc.b      'Dr. Dobb''s MandelZoom'

.ALIGN 2

MandelWindStorage   dcb.b     156, 0
ParamsDLOGStorage   dcb.b     170, 0
LegendDLOGStorage   dcb.b     170, 0

ItemHit             dc.w      0
ItemType            dc.w      0           ; Not Used
ItemHandle          dc.l      0           ; passed from _GetDItem to _GetIText
ItemBox             dcb.l     2, 0        ; Not Used
theString dcb.b     256, 0
theDialog dc.l      0           ; for dialogPtr returned by _IsDialogEvent

X_Org_Str dcb.b     10, 0
Y_Org_Str dcb.b     10, 0
Side_Length         dcb.b     10, 0

Count_Strings       dcb.b     40, 0       ; 4 X 10 Bytes each
TempRect            dcb.l     2, 0        ; holds the Patt Rects for the Legend
TempSTR             dcb.b     40, 0
EventRecord
   What:            dc.w      0
   Message:         dc.l      0
   When:            dc.l      0
   Where: dc.l      0
   Modifiers:       dc.w      0

Patterns
        dc.l        $FFFFFFFF ; 4 pixels per 4 = black
        dc.l        $FFFFFFFF

        dc.l        $FFAAFFAA ; 3 pixels per 4 = dark gray
        dc.l        $FFAAFFAA

        dc.l        $AA00AA00 ; 1 pixel  per 4 = light gray
        dc.l        $AA00AA00

        dc.l        $00000000 ; 0 pixels per 4 = pure white
        dc.l        $00000000

        dc.l        $AA55AA55 ; 2 pixels per 4 = gray
        dc.l        $AA55AA55

; ---------------- Variables ( off A5 ) -------------------

X_Origin            ds.l      1           ; Fixed-Pt conversions from
Y_Origin            ds.l      1           ; User Entries in Params DLOG
X_Side              ds.l      1
Y_Side              ds.l      1           ; Set to X_Side for now

Y_Start             ds        1           ; where Pen first Plots

Num_Rows            ds        1
Num_Cols            ds        1
Pix_Per_Pt          ds        1

Counts              ds.w      4           ; 4 INTEGERs dividing the Iterative
                                          ; Domain into 5 Ranges (& Patterns)
Iter_Count          ds.w      1
Row_Count ds.w      1
Col_Count ds.w      1

y_current ds.w      1
x_current ds.w      1

C_Real              ds.w      1
                    ds.w      1
C_imag              ds.w      1
                    ds.w      1

Z_Real              ds.w      1
                    ds.w      1
Z_imag              ds.w      1
                    ds.w      1

Del_C_Real          ds.l      1
Del_C_imag          ds.l      1

Legend_Y_Pos        ds.w      1

Start_Time          ds.l      1

Radio_1_State       ds.b      1
Radio_2_State       ds.b      1
Radio_3_State       ds.b      1

First_Entry         ds.b      1
Mouse_Down          ds.b      1
First_Pt            ds.b      1

        END
```

End Listing One

Listing Two

```
Listing Two

; < Str2FP.INC >    Thur 10 April '86   h. katz
;                        Mon  14 April '86

; This File is Linked with MandelZoom.ASM to provide String-to Floating-Point
; and Floating-Point to Fixed-Point Conversions for the
; X_Org, Y_Org, and Side_Len DITL Parameters

; At present only Single-Precision SANE Conversions are used

; A2 = ptr to the Decimal String on Input
; D0 = the Fixed-Pt Number for Output
String_Format 3

Include MacTraps.D
Include SANEMacs.Txt

XDEF      Convert_2_Fixed_Point

Sign                EQU       0         ; Byte Offsets in Decimal Record
Exp                 EQU       2
Sig                 EQU       4

FP_Sign             EQU       31        ; Bit Offsets in Single-Precision Result
FP_Exp              EQU       30
FP_Sig              EQU       22

SP_Exp_Bias         EQU       127
DP_Exp_Bias         EQU       1023      ; Code for Double-Precision not written

Convert_2_Fixed_Point

          lea       Temp_String, a0             ; make a copy of incoming string
          move.b    (a2), d0          ; its length
@0        move.b    (a2)+, (a0)+
          DBRA      d0, @0
          lea       Temp_String, a0             ; replace ptr to theString

          BSR       Build_Decimal_Record

          pea       Decimal_Record
          pea       FP_Num
          FDEC2S                                ; a SANE 'trap'

          BSR       Build_Fixed_Pt

          RTS                                   ; to Mandelbrot

Build_Fixed_Pt

          sf        d2                          ; assume Positive
          lea       FP_Num, a0
          move.l    (a0), d1          ; save the Exponent
          lsl.l     #1, d1                      ; shift Sign into Carry
          bcc       @1                          ; was Positve
          st        d2                          ; flag as Negative

@1        swap      d1                          ; move Exp into Low Word
          lsr.w     #8, d1                      ; shift Exp to Right
          sub.b     #SP_Exp_Bias, d1  ; unbias it

          move.l    (a0), d0          ; move orig FP_Num into register
          and.l     #$007FFFFF, d0              ; clear Exp
          bset.l    #23, d0                     ; add the leading '1' bit
          tst.b     d2
          beq       @2                          ; num is +
          neg.l     d0

@2        sub.b     #7, d1                      ; Neg(7-Exp) is amount to shift
          neg.b     d1
          bmi       @Shift_Left
          asr.l     d1, d0
          bra       @rts

@Shift_Left
          neg.b     d1                ; Max Left Shift not checked for yet
          asl.l     d1, d0

@rts      RTS

Build_Decimal_Record

          ; Strip the Sign Char
          ; Strip the Decimal_Pt and Decrease the Exponent Accordingly
          ; Finally Strip Leading Zeroes

          lea       Decimal_Record, a1 ; Zero the Record
          move.l    #0, (a1)+
          move.l    #0, (a1)+
          move.l    #0, (a1)+
          move.l    #0, (a1)+
          lea       Decimal_Record, a1

          cmp.b     #'+', 1(a0)                 ; Strip the Plus Sign, if any
          BNE       @Strip_Minus_Sign
          BSR       Shift_Count_Byte
          bra       Strip_Decimal_Pt

@Strip_Minus_Sign
          cmp.b     #'-', 1(a0)
          BNE       Strip_Decimal_Pt
          move.b    #1, Sign(a1)                ; Mark Dec_Rec Sign as Negative
          BSR       Shift_Count_Byte

Strip_Decimal_Pt

          move.b    (a0), Sig(a1)               ; move Count to Decimal_Record
```

(continued on page 86)

dBASE and Pascal. Together at last.

It had to happen.

They've been going together for years, but always with a lot of "glue, tape and hope" procedures that kept it from being a perfect union.

Announcing a new dBASE® enhancement tool that includes an interface to Turbo Pascal™ and a Pascal library. So you can customize the dBASE standard to your special business and scientific/engineering needs.

Array handling.

Random Number Generation function.

Variance and coefficient of variance.

Standard deviation function.

Skewness and sample kurtosis.

And business functions like present and future value, net present value, loan amortization, and internal rate of return.

Plus you'll be able to write and store custom functions in the library.

To order, or for more information or the name of your nearest dealer, call the Ashton-Tate® Publishing Group at (800) 437-4329, Ext. 227.

And live happily ever after.

Circle **no. 242** on reader service card.

Listing Two (Listing continued, text begins on page 42.)

```
        lea      1+Sig(a1), a2          ; point to 1st Digit
        add.l    #1, a0                 ; points to 1st Digit in Src Str
        clr      d0
        move.b   Sig(a1), d0            ; length of String
        sub.b    #1, d0                 ; - 1 for DBRA
        sf       d1                     ; Passed_Decimal_Pt Flag = FALSE

@0      cmp.b    #'.', (a0)
        beq      @Found_Decimal_Pt
        move.b   (a0)+, (a2)+           ; shift the digit to Decimal_Rec
        tst.b    d1                     ; are we past the Decimal_Pt ?
        beq      @Test_EOStr
        sub      #1, Exp(a1)
        bra      @Test_EOStr

@Found_Decimal_Pt

        st       d1
        add.l    #1, a0                 ; point past decimal point
        sub.b    #1, Sig(a1)            ; Count := Count - 1

@Test_EOStr

        DBRA     d0, @0

Strip_Leading_Zeroes

        lea      Sig(a1), a0            ; point @ Count Byte in
        move.b   (a0), d0      ; Decimal_Record Sig Field
        sub.b    #1, d0                 ; setup for DBRA
        ext.w    d0
        sf       d1                     ; No Leading Zeroes (yet)

@Loop_1
        cmp.b    #'0', 1(a0)
        BNE      @Test_4_Shift          ; Encountered a Signif Digit -> Done
        st       d1
        BSR      Shift_Count_Byte
        DBRA     d0, @Loop_1

@Test_4_Shift

        tst.b    d1            ; Any Non-Significant Zeroes Found ?
        BEQ      @rts          ; no

        lea      Sig(a1), a1   ; point to Sig Count Byte
        move.b   (a0), d0  ; a0 is Count Byte (wherever it is)
        ext.w    d0

@Loop_2
          move.b   (a0)+, (a1)+        ; shift Count + Digits to Left
          DBRA     d0, @Loop_2

@rts      RTS

Shift_Count_Byte

          sub.b    #1, (a0)  ; Length = Length - 1
          move.b   (a0)+, (a0)         ; move Count Byte over one
          RTS

; -------------------------------------------------------

FP_Num              dc.b     8, 0      ; working space for both
                                       ; Single & Double Precision numbers
Decimal_Record
                    dc.w      0        ; Sign
                    dc.w      0        ; Exp
                    dcb.b    12, 0     ; Sig

Temp_String         dcb.b    12, 0

END
```

End Listing Two

Listing Three

```
*  ManDLOG.R         Mon 31 April '86  h. katz
*
*
* Output file -

Mandels:Mandel_2
APPLKATZ

* Input file -
Include Mandels:Mandel_2.CodeR


Type WIND              ;; Mandelbrot Window
          ,101

41 6 336 404
Visible NoGoAway
0                           ;; docProc
0


Type DLOG              ;;'Legend' Dialog
          ,101
no message
30 415 330 500
Visible NoGoAway
1                           ;; DBoxProc
0
101


Type DITL
          ,101
2                           ;; 2 Items

Button                      ;; Item #1
240 3 265 83
New Plot

Button                      ;; Item #2
270 3 295 83
Quit


Type DLOG              ;; Parameters Dialog
          ,100
no message
50 100 250 400
Visible NoGoAway
1                           ;; dBoxProc
0
100


Type DITL
          ,100
18                          ;; 18 Items

Button                      ;; Item #1
135 130 160 220
Plot

Button                      ;; Item #2
165 130 190 220
Quit

StaticText Disabled  ;; Item #3
8 30 25 235
Mandelbrot Parameters

StaticText Disabled  ;; Item #4
40 15 56 75
X_Origin

StaticText Disabled  ;; Item #5
70 15 86 75
Y_Origin

StaticText Disabled  ;; Item #6
100 15 116 75
    Side

StaticText Disabled  ;; Item #7
65 180 80 240
Counts

StaticText Disabled  ;; Item #8
155 15 170 45
Pen

RadioButton                   ;; Item #9
135 50 150 105
1 X 1

RadioButton                   ;; Item #10
155 50 170 105
2 X 2

RadioButton                   ;; Item #11
175 50 190 105
4 X 4

EditText             ;; Item #12          X_Origin
40 80 55 144
-2.00

EditText             ;; Item #13          Y_Origin
70 80 85 144
-1.25

EditText             ;; Item #14          Side_Length
100 80 115 144
2.500

EditText             ;; Item #15          4 initial Defaults for
                                                      Patt Ranges
20 245 35 280
32

EditText             ;; Item #16
50 245 65 280
12

EditText             ;; Item #17
80 245 95 280
6

EditText             ;; Item #18
110 245 125 280
4


TYPE ALRT
          ,1
40 100 180 400
1
7777                  ;; Default - Item 1 / Draw Box / 3 Beeps
                                             ( ALL stages )
                      ;;        0       /   1    /   10

TYPE DITL
          ,1
2

Button
100 220 120 270
OK

StaticText Disabled
40 30 60 290
Numeric Digits Only
```

End Listings

DIGITAL DISSOLVE

Listing One (Text begins on page 48.)

```
;
;  procedure DissBits (srcB, destB: bitMap; srcR, dstR: rect); external;
;
; mike morton
; release: 30 june 1986, version 5.3
; this version is formatted for the Lisa Workshop assembler
;
; differences from version 5.2:
;        extraneous code removed from bitwidth routine
;        introductory comments are much shorter
;
; ***********************************************************************
; *       copyright 1984, 1985, 1986 by michael s. morton               *
; ***********************************************************************
;
; DissBits is freeware.  you're welcome to copy it, use it in programs, and
; to modify it, as long as you leave my name in it.  i'd be interested in
; seeing your changes, especially if you find ways to make the central
; loops faster, or port it to other machines/languages.
;
; if, for some reason, you only have a hard copy of this and would like a
; source on a diskette, please contact:
;       robert hafer
;       the boston computer society
;       one center plaza
;       boston, mass.  02108

;
; include files:
;       tlasm/graftypes -- definitions of "bitMap" and "rect"
;       tlasm/quickmacs -- macros for quickdraw calls (e.g., _hidecursor)
;


.nolist
.include tlasm/graftypes
.include tlasm/quickmacs
.list

;
; definitions of the "ours" record: this structure, of which there are
; two copies in our stack frame, is a sort of bitmap:
;

oRows   .equ      0                  ; (word) number of last row (first is 0)
oCols   .equ      oRows+2            ; (word) number of last column (first is 0)
oLbits  .equ      oCols+2            ; (word) size of left margin within 1st byte
oStride .equ      oLbits+2 ; (word) stride in memory from row to row
oBase   .equ      oStride+2 ; (long) base address of bitmap

osize   .equ      oBase+4            ; size, in bytes, of "ours" record

;
; stack frame elements:
;

srcOurs .equ      -osize             ; (osize) our view of source bits
dstOurs .equ      srcOurs-osize      ; (osize) our view of target bits

sflast  .equ      dstOurs            ; relative address of last s.f. member
sfsize  .equ      -sflast            ; size of s.f. for LINK (must be EVEN!)
;
;       parameter offsets from the stack frame pointer, A6:
;       last parameter is above return address and old s.f.
;

dRptr   .equ      4+4                ; ^destination rectangle
sRptr   .equ      dRptr+4            ; ^source rectangle
dBptr   .equ      sRptr+4            ; ^destination bitMap
sBptr   .equ      dBptr+4            ; ^source bitMap

plast   .equ      sBptr+4            ; address just past last parameter

psize   .equ      plast-dRptr        ; size of parameters, in bytes

;
; entrance: set up a stack frame, save some registers, hide the cursor.
;

.proc   dissBits             ; main entry point

        link     A6,#-sfsize        ; set up a stack frame
        movem.l  D3-D7/A2-A5,-(SP)        ; save registers compiler may need
        _hidecurs           ; don't let the cursor show for now

;
; convert source and destination bitmaps and rectangles to a format we prefer.
; we won't look at these parameters after this.
;

        move.l   sBptr(A6),A0       ; point to source bitMap
        move.l   sRptr(A6),A1       ; and source rectangle
```

(continued on page 90)

Techical Product Information.

Dr. Dobb's Journal of Software Tools

Name ____________________

Title ____________________

Company ____________________ Phone ____________________

Address ____________________

City/State/Zip ____________________

November 1986 #121 Expiration Date: February 28, 1987

Please circle one letter in each category:

I. My work is performed:
- A. for in-house use only.
- B. for other companies.
- C. for end users/retailers.
- D. in none of the above areas.

II. My primary job function:
- A. Software Project Mgmt/Spvr
- B. Hardware Project Mgmt/Spvr
- C. Computer Consultant
- D. Corporate Consultant
- E. Other

III. My company department performs:
- A. software development.
- B. computer system integration.
- C. computer manufacturing.
- D. computer consulting.
- E. computer research
- F. none of the above.

IV. This inquiry is for:
- A. a purchase within 1 month.
- B. a purchase within 1 to 6 months.
- C. product information only.

V. Corporate Purchase Authority:
- A. Final Decision-maker
- B. Approve/Recommend
- C. No Influence

VI. Personal Computer Users at my Jobsite:
- A. 10,000 or more
- B. 500 to 9,999
- C. 100 to 499
- D. 10 to 99
- E. less than 10

VII. On average, I advise others about computers:
- A. more than once per day.
- B. once per day.
- C. once per week.
- D. less than once per week.

VIII. In my job function, I:
- A. design software and/or write code.
- B. design software.
- C. write code.
- D. don't design software or write code.

A Reader Service number appears on each advertisement. Circle the corresponding numbers at right for more info.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290
291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310
311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330
331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370
371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390
391	392	393	394	395	396	397	398	399	999

Circle 999 to start a 12 month subscription at the price of $29.97

TAKE THIS CARD WITH YOU AS YOU READ THROUGH THIS ISSUE OF DR. DOBB'S.

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

First Class Permit #217, Clinton, Iowa

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

Technical Product Information.

TAKE THIS CARD WITH YOU AS YOU READ THROUGH THIS ISSUE OF DR. DOBB'S.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290
291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310
311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330
331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370
371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390
391	392	393	394	395	396	397	398	399	999

Circle 999 to start a 12 month subscription at the price of $29.97

Dr. Dobb's Journal of Software Tools

Name ____________________

Title ____________________

Company ____________________ Phone ____________________

Address ____________________

City/State/Zip ____________________

November 1986 #121 Expiration Date: February 28, 1987

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Consultant
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

A Reader Service number appears on each advertisement. Circle the corresponding numbers at left for more info.

NEW! FROM BLAISE COMPUTING

C Tools Plus™

For The Programmer Whose Alphabet Begins & Ends With "C"

Today's programmers need more than yesterday's tools. Requirements such as removable windows and "sidekickable" pop-up utilities are changing the face of program design. You need to filter interrupts so that other resident programs still work. You need the ability to switch between multiple display pages and monitors. Today's technical demands are almost endless, but C TOOLS PLUS gives you what you need.

SOLID LIBRARY SUPPORT

Blaise Computing offers you solid library support that can meet all your demands and more. C TOOLS PLUS embodies the full spectrum of general-purpose utility functions that are critical to today's applications.

Here's just part of the PLUS in C TOOLS PLUS:

- **C TOOLS and C TOOLS 2** compatibility—two packages that receive rave reviews for quality, organization, usability and documentation.
- **FULL SOURCE CODE**
- **WINDOWS** that are stackable, removable, that support word wrap and that can accept user input.
- **INTERRUPT SERVICE ROUTINE** support for truly flexible, robust and polite resident applications.
- **MULTIPLE** monitor and display support, including EGA 43-line mode.
- **FAST DIRECT VIDEO ACCESS** for efficiency that will not constrain good program design.
- **DOCUMENTATION, TECHNICAL SUPPORT** and attention to detail that have distinguished Blaise Computing products over the years.

C TOOLS PLUS supports the Microsoft (and IBM) 3.00 and Lattice 3.00 C compilers and is just ***$175.00.***

Also Available Are:

C VIEW MANAGER— *A kit for building data entry screens and menus. Begin by designing on-screen what the operator will see; call upon our library functions from your program to display the screens and retrieve the data. Just $275, including all library source code.*

C ASYNCH MANAGER — *provides the crucial core of hardware interrupt support needed to build applications that communicate. It also includes the "XMODEM" file-transfer protocol and support for Hayes-compatible modems. All source code is included for $175.*

C TOOLS & C TOOLS 2—*an indispensable combination still available at a low price of $175, including all source code. See review in* PC Tech *Journal, 6/85.*

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

ORDER TOLL-FREE 800-227-8087!

YES, send me the PLUS I need! Enclosed is $______ for C TOOLS PLUS. (CA residents add 6½% Sales Tax. All domestic orders add $10.00 for Federal Express shipping.)

Name: ______ Phone: (___) ______

Shipping Address: ______

City: ______ State: ____ Zip: ____

VISA or MC #: ______ Exp. Date: ____

Circle **no. 159** on reader service card.

DIGITAL DISSOLVE

Listing One (Listing continued, text begins on page 48.)

```
        lea     srcOurs(A6),A2          ; and our source structure
        bsr     CONVERT                 ; convert to our format

        move.l  dBptr(A6),A0            ; point to destination bitMap
        move.l  dRptr(A6),A1            ; and rectangle
        lea     dstOurs(A6),A2          ; and our structure
        bsr     CONVERT                 ; convert to our format

;
; check that the rectangles match in size.
;
        move.w  srcOurs+oRows(A6),D0        ; pick up the number of rows
        cmp.w   dstOurs+oRows(A6),D0          ; same number of rows?
        bne     ERROR                   ; nope -- bag it

        move.w  srcOurs+oCols(A6),D0        ; check the number of columns
        cmp.w   dstOurs+oCols(A6),D0          ; same number of columns, too?
        bne     ERROR                   ; that's a bozo no-no

;
; figure the bit-width needed to span the columns, and the rows.
;

        move.w  srcOurs+oCols(A6),D0        ; get count of columns
        ext.l   D0                      ; make it a longword
        bsr     LOG2                    ; figure bit-width
        move.w  D0,D1                   ; set aside that result
        beq     SMALL                   ; too small?  wimp out and use copyBits

        move.w  srcOurs+oRows(A6),D0        ; get count of rows
        ext.l   D0                      ; make it a longword
        bsr     LOG2                    ; again, find the bit-width
        tst.w   D0                      ; is the result zero?
        beq     SMALL                   ; if so, our algorithm will screw up

;
; set up various constants we'll need in the in the innermost loop
;

        move.l  #1,D5                   ; set up...
        lsl.l   D1,D5                   ; ...the bit mask which is...
        sub.l   #1,D5                   ; ...bit-width (cols) 1's

        add.w   D1,D0                   ; find total bit-width (rows plus columns)
        lea     TABLE,A0  ; point to the table of XOR masks
        moveq   #0,D3                   ; clear out D3 before we fill the low byte
        move.b  0(A0,D0),D3             ; grab the correct XOR mask in D3

;
; table is saved compactly, since no mask is wider than a byte.
;  we have to unpack it so high-order bit of the D0-bit-wide field is on:
;

UNPACK  add.l   D3,D3                   ; shift left by one
        bpl.s   UNPACK                  ; keep moving until top bit that's on is
                                        ; aligned at the top end

        rol.l   D0,D3                   ; now swing the top D0 bits around to be
                                        ; bottom D0 bits, the mask
        move.l  D3,D0                   ; 1st sequence element is the mask itself

;
; do all kinds of preparation:
;

        move.l  srcOurs+oBase(A6),D2        ; set up base ptr for source bits
        lsl.l   #3,D2                   ; make it into a bit address
        move.l  D2,A0                   ; put it where the fast loop will use it
        move.w  srcOurs+oLbits(A6),D2       ; now pick up source left margin
        ext.l   D2                      ; make it a longword
        add.l   D2,A0                   ; make A0 useful for odd routine below

        move.l  dstOurs+oBase(A6),D2        ; set up base pointer for target
        lsl.l   #3,D2                   ; again, bit addressing works out faster
        move.l  D2,A1                   ; stuff it where we want it for the loop
        move.w  dstOurs+oLbits(A6),D2       ; now pick up destination left margin
        ext.l   D2                      ; make it a longword
        add.l   D2,A1                   ; and make A1 useful, too

        move.w  srcOurs+oCols(A6),A2        ; pick up the often-used count
                                              ; of columns
        move.w  srcOurs+oRows(A6),D2        ; and of rows
        add.w   #1,D2                   ; make row count one-too-high for compares
        ext.l   D2                      ; and make it a longword
        lsl.l   D1,D2                   ; slide it to line up w/rows part of D0
        move.l  D2,A4                   ; and save that somewhere useful

        move.w  D1,D2                   ; put log2(columns) in a safe place (sigh)
```

(continued on page 92)

DIGITAL DISSOLVE

Listing One (Listing continued, text begins on page 48.)

```
;
; try to reduce the amount we shift down D2.  this involves:
;     halving the strides as long as each is even, decrementing D2 as we go
;     masking the bottom bits off D4 when we extract the row count in the loop
;
; alas, can't always shift as little as we want.  for instance, if we don't
; shift down far enough, row count will be so high as to exceed a halfword,
; and the dread mulu instruction won't work (eats only word operands).  so,
; we have to have an extra check to take us out of the loop early.
;

          move.w    srcOurs+oStride(A6),D4   ; pick up source stride
          move.w    dstOurs+oStride(A6),D7   ; and target stride
          move.w    srcOurs+oRows(A6),D1     ; get row count for klugey check

          tst.w     D2                      ; how's the bitcount?
          beq.s     HALFDONE             ; skip out if already down to zero

HALFLOOP
          btst      #0,D4                ; is this stride even?
          bne.s     HALFDONE  ; nope -- our work here is done
          btst      #0,D7                ; how about this one?
          bne.s     HALFDONE  ; have to have both even

          lsl.w     #1,D1                ; can we keep max row number in a halfword?
          bcs.s     HALFDONE  ; nope -- D2 mustn't get any smaller!

          lsr.w     #1,D4                ; halve each stride...
          lsr.w     #1,D7                ; ...like this
          sub.w     #1,D2                ; and remember not to shift down as far
          bne.s     HALFLOOP  ; loop unless we're down to no shift at all

HALFDONE                       ; no tacky platitudes, please
          move.w    D4,srcOurs+oStride(A6)   ; put back source stride
          move.w    D7,dstOurs+oStride(A6)   ; and target stride

;
; make some stuff faster to access -- use the fact that (An) is faster
; to access than d(An).  this means we'll misuse our frame pointer, but
; don't worry -- we'll restore it before we use it again.
;

          move.w    srcOurs+oStride(A6),A5  ; make source stride faster
                                            ; to access, too
          move.l    A6,-(SP)             ; save framitz pointer
          move.w    dstOurs+oStride(A6),A6  ; pick up destination stride
          move.l    #0,D6                ; we do only AND.w x,D6 -- but ADD.l D6,x

          clr.w     -(SP)                ; reserve room for function result
          bsr       MULCHK               ; go see if strides are powers of two
          tst.w     (SP)+                ; can we eliminate the horrible MULUs?
          bne       NOMUL                ; yes!  hurray!

;
; main loop: map the sequence element into rows and columns, check if it's
; in bounds and skip on if it's not, flip the appropriate bit, generate
; the next element in the sequence, and loop if the sequence isn't done.
;

;
; check row bounds. note that we can check row before extracting it from
; D0, ignoring bits at bottom of D0 for the columns. to get these bits
; to be ignored, had to make A4 1-too-high before shifting up to align it.
;

LOOP                                     ; here for another time around
          cmp.l     A4,D0                ; is row in bounds?
          bge.s     NEXT                 ; no: clip this

;
; map it into the column; check bounds.          note that we save this check
; for second; it's a little slower because of the move and mask.
;
; chuck sagely points out that when the "bhi" at the end of the loop takes, we
; know we can ignore the above comparison.  thanks, chuck.  you're a
; great guy.
;

LOOPROW                                  ; here when we know the row number is OK
          move.w    D0,D6                ; copy the sequence element
          and.w     D5,D6                ; find just the column number

          cmp.w     A2,D6                ; too far to the right? (past oCols?)
          bgt.s     NEXT                 ; yes: skip out

          move.l    D0,D4                ; we know element will be used; copy it
          sub.w     D6,D4                ; remove column's bits
          lsr.l     D2,D4                ; shift down to row, NOT right-justified
```

(continued on page 94)

Circle **no. 286** on reader service card.

Circle **no. 285** on reader service card.

Circle **no. 294** on reader service card.

DIGITAL DISSOLVE

Listing One **(Listing continued, text begins on page 48.)**

```
;
; get source byte, and bit offset.  D4 has the bit offset in rows, and
; D6 is columns.
;

        move.w  A5,D1      ; get the stride per row (in bits)
        mulu    D4,D1      ; stride * row; find source row's offset in bits
        add.l   D6,D1      ; add in column offset (bits)
        add.l   A0,D1      ; plus base of bitmap (bits [sic])
        move.b  D1,D7      ; save the bottom three bits for the BTST
        lsr.l   #3,D1      ; while we shift down to a word address
        move.l  D1,A3      ; and save that for the test, too
        not.b   D7         ; get right bit number (compute #7-D7)

;
; find the destination bit address and bit offset
;

        move.w  A6,D1      ; extract cunningly hidden destination stride
        mulu    D1,D4      ; stride*row number = dest row's offset in bits
        add.l   D6,D4      ; add in column bit offset
        add.l   A1,D4      ; and base address, also in bits
        move.b  D4,D6      ; set aside the bit displacement
        lsr.l   #3,D4      ; make a byte displacement
        not.b   D6         ; get right bit number (compute #7-D6)

        btst    D7,(A3)    ; test the D7th bit of source byte
        move.l  D4,A3      ; point to target byte (don't lose CC from btst)
        bne.s   SETON      ; if on, go set destination on
        bclr    D6,(A3)    ; else clear destination bit

;
; find the next sequence element.  see knuth, vol ii., page 29
; for sketchy details.
;

NEXT                       ; jump here if D0 not in bounds
        lsr.l   #1,D0      ; slide one bit to the right
        bhi.s   LOOPROW    ; if no carry out, but not zero, loop

        eor.l   D3,D0      ; flip magic bits for bitwidth we want...
        cmp.l   D3,D0      ; ...but has this brought us to square 1?
        bne.s   LOOP       ; if not, loop back; else...

        bra.s   DONE       ; ...we're finished

SETON
        bset    D6,(A3)    ; source bit was on: set destination on

        ; copy of above code, stolen for inline speed -- sorry.
        lsr.l   #1,D0      ; slide one bit to the right
        bhi.s   LOOPROW    ; if no carry out, but not zero, loop
        eor.l   D3,D0      ; flip magic bits...
        cmp.l   D3,D0      ; ...but has this brought us to square 1?
        bne.s   LOOP       ; if not, loop back; else fall through


;
; here when done; the (0,0) point has not been done yet.  this is
; really the (0,left margin) point. also jump here from another copy loop.
;

DONE
        move.l  (SP)+,A6   ; restore stack frame pointer

        move.w  srcOurs+oLbits(A6),D0    ; pick up bit offset of left margin
        move.w  dstOurs+oLbits(A6),D1    ; and ditto for target
        not.b   D0                      ; flip to number the bits for 68000
        not.b   D1                      ; ditto

; alternate, late entrance, when SCREEN routine has already set up D0 and
; D1 (it doesn't want the bit offset negated).

DONEA                                   ; land here with D0, D1 set
        move.l  srcOurs+oBase(A6),A0     ; set up base ptr for source bits
        move.l  dstOurs+oBase(A6),A1     ; and pointer for target

        bset    D1,(A1)                 ; assume source bit was on; set target
        btst    D0,(A0)                 ; was first bit of source on?
        bne.s   DONE2                   ; yes: skip out
        bclr    D1,(A1)                 ; no: oops!  set it right, and fall through

;
; return
;

DONE2                                   ; here when we're really done
ERROR                                   ; we return silently on errors
        _showcurs          ; let's see this again
```

(continued on page 96)

Experience the Future...

Macro Assemblers/ Cross Assemblers for Series 3.0

Series 3 requires at least 256k of memory to run under MSDOS & CP/M86 and 54k TPA of memory to run CP/M80. Series 3 products have: Full Listing Control, Conditional Assembly & Built in Cross Reference. There is Unlimited Program Size, Number of Symbols and Macros. The Linkers output: Intel Hex, Extended Intel Hex, Tektronix Hex, and Motorola S19, S28, S37 formats.

Macro Assemblers/ Cross Assemblers for Series 4.0

Series 4 requires 512k of memory to run. These products have all the features of Series 3 plus: 32 Character Labels and User Defined Sections. There is one Linker on Series 4 which outputs all the same formats as Series 3. Series 4 runs under MSDOS, UNIX, VMS and ULTRIX.

Simulator-Debuggers

The Simulator-Debuggers run with 256k of memory. The Simulator has 16 Breakpoints with optional Counter Field. The Symbol Table is limited only by the amount of memory. Buffers of 256 bytes may be opened for I/O ports. The Simulators are available for MSDOS, UNIX, VMS and ULTRIX.

To order, call toll free in U.S. (including HI, PR and VI):

1 800 843-8144

In Colorado: (303) 369-5001

For more details, ask for a free brochure.

	Macro Assemblers Cross Assemblers			*Simulator-Debuggers*	
	Series 3	**Series 4**			
Software Product	**CP/M 80 CP/M 86 MSDOS**	**MSDOS AT&T UNIX**	**VMS Zeus ULTRIX**	**MSDOS AT&T UNIX**	**VMS Zeus ULTRIX**
Super 8		199.50	995.00	149.50	499.50
Z-8	199.50	199.50	995.00	149.50	499.50
Z-80	199.50	199.50	995.00	149.50	499.50
Z-8000	299.50	299.50	1250.00		
1802	199.50				
6301	199.50	199.50	995.00	149.50	499.50
64180	199.50	199.50	995.00		
6501	199.50	199.50	995.00		
6502	199.50	199.50	995.00		
65c02	199.50	199.50	995.00		
65c816		299.50	1250.00		
6800,2,8	199.50	199.50	995.00	149.50	499.50
6801,3	199.50	199.50	995.00		
6804	199.50	199.50	995.00		
6805	199.50	199.50	995.00		
6809	199.50	199.50	995.00		
68c11	199.50	199.50	995.00		
68000,8,10	299.50	299.50	1250.00		
68020		399.50	1500.00		
8400/c00	199.50	199.50	995.00		
8044/51	199.50	199.50	995.00	149.50	499.50
80515		199.50	995.00		
8080	199.50	199.50	995.00	149.50	499.50
8085	199.50	199.50	995.00	149.50	499.50
8086/88	99.50	99.50	1250.00		
8096	199.50	199.50	1250.00		
80186/286	199.50	199.50	1250.00		
83c351		199.50	995.00		
8748	199.50	199.50	995.00	149.50	499.50
V20/30	199.50	199.50	1250.00		
Cops 400	199.50				
F8/3870	199.50				
NCR32	399.50				
NEC7500	199.50				
NSC800	199.50	199.50	995.00	149.50	499.50

more formats available

Name ______________________ (Please Print)

Company ______________________

Address ______________________

City ______________ State ________ Zip ________

Phone ______________________

Check ☐ COD (U.S. Only) ☐

MC/VISA # ______________________

Expiration Date ______________________

Product ______________ Operating System ______________

Series ______________ Amount $ ______________

Shipping $ ______________ Total $ ______________

Signature ______________________

(Shipping is $20.00 per unit for overseas orders. Toll Free number does not apply to overseas. 2500AD pays COD charges.)

2500AD SOFTWARE INC.

17200 East Ohio Drive
Aurora, Colorado 80017
(303) 369-5001

Listing One **(Listing continued, text begins on page 48.)**

```
        movem.l    (SP)+,D3-D7/A2-A5  ; restore lots of registers
        unlk       A6                 ; restore caller's stack frame pointer
        move.l     (SP)+,A0  ; pop return address
        add.l      #psize,SP ; unstack parameters
        jmp  (A0)            ; home to mother

;
; ----------------------------------------------------------------
;
; sleazo code for when we're asked to dissolve very small regions.  if
; either dimension of the rectangle is too small, we bag it and just
; delegate the problem to copyBits.  a possible problem with this is
; if someone decides to substitute us for the standard copyBits routine
; -- this case will become recursive...
;

SMALL                                 ; here when it's too small
        move.l     sBptr(A6),-(SP)    ; push args: source bitmap
        move.l     dBptr(A6),-(SP)    ;         destination bitmap
        move.l     sRptr(A6),-(SP)    ;         source rectangle
        move.l     dRptr(A6),-(SP)    ;         destination rectangle
        move.w     #srcCopy,-(SP)     ;         transfer mode -- source copy
        clr.l      -(SP)              ;         mask region -- NIL
        _copyBits            ; do the copy in quickdraw-land
        bra.s      DONE2              ; head for home

;
; ---------------------------------------------------------------------
;
; code identical to the usual loop, but A5 and A6 have been changed to
; shift counts. other than that, it's the same.  really it is!  well, no,
; wait a minute... because we don't have to worry about the word-size
; mulu operands, we can collapse the shifts and countershifts further
; as shown below:

NOMUL                        ; here for alternate version of loop
        tst.w      D2        ; is right shift zero?
        beq.s      NOMUL2    ; yes: can't do much more...
        cmp.w      #0,A5     ; how about one left shift (for source stride)?
        beq.s      NOMUL2    ; yes: ditto
        cmp.w      #0,A6     ; and the other left shift (destination stride)?
        beq.s      NOMUL2    ; yes: can't do much more...

        sub.w      #1,D2     ; all three...
        sub.w      #1,A5     ; ...are...
        sub.w      #1,A6     ; ...collapsible
        bra.s      NOMUL     ; go see if we can go further

;
; see if we can do the super-special-case loop, which basically is
; equivalent to any rectangle where the source and destination are
; both exactly the width of the Mac screen.
;

NOMUL2                       ; here when D2, A5, and A6 are all collapsed
        tst.w      D2        ; did this shift get down to zero?
        bne.s      NLOOP     ; no: skip to first kludged loop
        cmp.w      #0,A5     ; is this zero?
        bne.s      NLOOP     ; no: again, can't make further optimization
        cmp.w      #0,A6     ; how about this?
        bne.s      NLOOP     ; no: the best-laid plans of mice and men...
        cmp.w      A2,D5     ; is there no check on the column?
        bne.s      NLOOP     ; not a power-of-two columns; rats!

        move.w     A0,D6     ; grab the base address of the source
        and.b      #7,D6     ; select the low three bits
        bne.s      NLOOP     ; doesn't sit on a byte boundary; phooey

        move.w     A1,D6     ; now try the base of the destination
        and.b      #7,D6     ; and select its bit offset
        beq.s      SCREEN    ; yes!  do extra-special loop!

;
; fast, but not super-fast loop, used when both source and destination
; bitmaps have strides which are powers of two.
;

NLOOP                        ; here for another time around
        cmp.l      A4,D0     ; is row in bounds?
        bge.s      NNEXT     ; no: clip this

NLOOPROW             ; here when we know the row number is OK
        move.w     D0,D6     ; copy the sequence element
        and.w      D5,D6     ; find just the column number

        cmp.w      A2,D6     ; too far to the right? (past oCols?)
        bgt.s      NNEXT     ; yes: skip out

        move.l     D0,D4     ; we know element will be used; copy it
        sub.w      D6,D4     ; remove column's bits
        lsr.l      D2,D4     ; shift down to row, NOT right-justified
```

(continued on page 98)

Listing One (Listing continued, text begins on page 48.)

```
          move.w    A5,D7     ; get log2 of stride per row (in bits)
          move.l    D4,D1     ; make a working copy of the row number
          lsl.l     D7,D1     ; * stride/row is source row's offset in bits
          add.l     D6,D1     ; add in column offset (bits)
          add.l     A0,D1     ; plus base of bitmap (bits [sic])
          move.b    D1,D7     ; save the bottom three bits for the BTST
          lsr.l     #3,D1     ; while we shift down to a byte address
          move.l    D1,A3     ; and save that for the test, too
          not.b     D7        ; get right bit number (compute #7-D7)
          move.w    A6,D1     ; extract log2 of destination stride
          lsl.l     D1,D4     ; stride*row number = dest row's offset in bits
          add.l     D6,D4     ; add in column bit offset
          add.l     A1,D4     ; and base address, also in bits
          move.b    D4,D6     ; set aside the bit displacement
          lsr.l     #3,D4     ; make a byte displacement
          not.b     D6        ; get right bit number (compute #7-D6)

          btst      D7,(A3)   ; test the D7th bit of source byte
          move.l    D4,A3     ; point to target byte (don't ruin CC from btst)
          bne.s     NSETON    ; if on, go set destination on
          bclr      D6,(A3)   ; else clear destination bit

NNEXT                         ; jump here if D0 not in bounds
          lsr.l     #1,D0     ; slide one bit to the right
          bhi.s     NLOOPROW  ; if no carry out, but not zero, loop
          eor.l     D3,D0     ; flip magic bits...
          cmp.l     D3,D0     ; ...but has this brought us to square 1?
          bne.s     NLOOP     ; if not, loop back; else...
          bra.s     DONE      ; ...we're finished

NSETON
          bset      D6,(A3)   ; source bit was on: set destination on
          lsr.l     #1,D0     ; slide one bit to the right
          bhi.s     NLOOPROW  ; if no carry out, but not zero, loop
          eor.l     D3,D0     ; flip magic bits...
          cmp.l     D3,D0     ; ...but has this brought us to square 1?
          bne.s     NLOOP     ; if not, loop back; else fall through
          bra.s     DONE      ; and finish

;
; ----------------------------------------------------------------------
;
; super-special case, which happens to hold for the whole mac screen --
; or subsets of it which are as wide as the screen. here, we've found that
; the shift counts in D2, A5, and A6 can all be collapsed to zero.
; and D5 equals A2, so there's no need to check whether D6 is in limits --
; or even take it out of D0! so, this loop is the NLOOP code without
; the shifts or the check on the column number. should run like a bat;
; have you ever seen a bat run?
;
; one further restriction -- the addresses in A0 and A1 must point to
; integral byte addresses with no bit offset. (this still holds
; for full-screen copies.) because both the source and destination are
; byte-aligned, we can skip the ritual Negation Of The Bit Offset which
; the 68000 usually demands.

SCREEN    ; here to set up to do the whole screen, or at least its width
          move.l    A0,D6     ; take the base source address...
          lsr.l     #3,D6     ; ... and make it a byte address
          move.l    D6,A0     ; replace pointer

          move.l    A1,D6     ; now do the same...
          lsr.l     #3,D6     ; ...for...
          move.l    D6,A1     ; ...the destination address

          bra.s     N2LOOP    ; jump into loop

N2HEAD                        ; here when we shifted and a bit carried out
          eor.l     D3,D0     ; flip magic bits to make the sequence work

N2LOOP                        ; here for another time around
          cmp.l     A4,D0     ; is row in bounds?
          bge.s     N2NEXT    ; no: clip this

N2LOOPROW           ; here when we know the row number is OK
          move.l    D0,D1     ; copy row number, shifted up, plus column offset
          lsr.l     #3,D1     ; while we shift down to a word offset

          btst      D0,0(A0,D1)         ; test bit of source byte
          bne.s     N2SETON             ; if on, go set destination on
          bclr      D0,0(A1,D1)         ; else clear destination bit

N2NEXT                                  ; jump here if D0 not in bounds
          lsr.l     #1,D0               ; slide one bit to the right
          bhi.s     N2LOOPROW ; if no carry out, but not zero, loop
          bne.s     N2HEAD              ; if carry out, but not zero, loop earlier
          bra.s     N2DONE    ; 0 means next sequence element would have been D3

N2SETON
```

(continued on page 100)

Listing One (Listing continued, text begins on page 48.)

```
        bset      D0,0(A1,D1)         ; source bit was on: set destination on
        lsr.l     #1,D0               ; slide one bit to the right
        bhi.s     N2LOOPROW ; if no carry out, but not zero, loop
        bne.s     N2HEAD              ; if carry out, but not zero, loop earlier
                                      ; zero means the loop has closed on itself

;
; because our bit-numbering isn't like that of the other two loops, we set
; up D0 and D1 ourselves before joining a bit late with the common code to
; get the last bit.
;
N2DONE
        move.l    (SP)+,A6  ; restore the stack frame pointer

        move.w    srcOurs+oLbits(A6),D0    ; get bit offset of left margin
        move.w    dstOurs+oLbits(A6),D1    ; and ditto for target
        bra       DONEA     ; go do first bit, which sequence doesn't cover

;
; ----------------------------------------------------------------------
;

; mulchk -- see if we can do without multiply instructions.
;
; calling sequence:
;       A5 holds the source stride
;       A6 holds the destination stride
;       clr.w     -(SP)               ; reserve room for boolean function return
;       bsr       MULCHK              ; go check things out
;       tst.w     (SP)+               ; test result
;       bne.s     SHIFT               ; if non-zero, we can shift and not multiply
;
;       (if we can shift, A5 and A6 have been turned into shift counts)
;
; registers used: none  (A5, A6)

MULCHK

        movem.l   D0-D3,-(SP)         ; stack caller's registers
        move.l    A5,D0               ; take the source stride
        bsr       BITWIDTH  ; take log base 2
        move.l    #1,D1               ; pick up a one...
        lsl.l     D0,D1               ; ...and try to recreate the stride
        cmp.l     A5,D1               ; does it come out the same?
        bne.s     NOMULCHK  ; nope -- bag it
        move.w    D0,D3               ; save magic logarithm of source stride

        move.l    A6,D0               ; yes -- now how about destination stride?
        bsr       BITWIDTH  ; convert that one, also
        move.l    #1,D1               ; again, try a single bit...
        lsl.l     D0,D1               ; ...and see if original # was 1 bit
        cmp.l     A6,D1               ; how'd it come out?
        bne.s     NOMULCHK  ; doesn't match -- bag this

;
; we can shift instead of multiplying.  change address registers & tell
; our caller.
;
        move.w    D3,A5               ; set up shift for source stride
        move.w    D0,A6               ; and for destination stride
        st        4+16(SP)  ; tell our caller what's what
        bra.s     MULRET              ; and return

NOMULCHK

        sf 4+16(SP)                   ; tell caller we can't optimize
MULRET                                ; here to return; result set
        movem.l   (SP)+,D0-D3         ; pop some registers
        rts                           ; all set

;
; ----------------------------------------------------------------------
;
; table of (longword) masks to XOR in strange Knuthian algorithm.
; the first table entry is for a bit-width of two, so the table actually
; starts two bytes before that.        hardware jocks among you may recognize
; this scheme as the software analog of a "maximum-length sequence
; generator".
;
; to save a bit of room, masks are packed in bytes, but should be aligned
; as described in the code before being used.
;

table   .equ      *-2                 ; first element is #2
        .byte     3o                  ; 2
        .byte     3o                  ; 3
        .byte     3o                  ; 4
        .byte     5o                  ; 5
        .byte     3o                  ; 6
```

(continued on page 102)

Listing One (Listing continued, text begins on page 48.)

```
        .byte   3o              ; 7
        .byte   27o             ; 8
        .byte   21o             ; 9
        .byte   11o             ; 10
        .byte   5o              ; 11
        .byte   145o            ; 12
        .byte   33o             ; 13
        .byte   65o             ; 14
        .byte   3o              ; 15
        .byte   55o             ; 16
        .byte   11o             ; 17
        .byte   201o            ; 18
        .byte   71o             ; 19
        .byte   11o             ; 20
        .byte   5o              ; 21
        .byte   3o              ; 22
        .byte   41o             ; 23
        .byte   33o             ; 24
        .byte   11o             ; 25
        .byte   161o            ; 26
        .byte   71o             ; 27
        .byte   11o             ; 28
        .byte   5o              ; 29
        .byte   145o            ; 30
        .byte   11o             ; 31
        .byte   243o            ; 32
.align 2
;
; -------------------------------------------------------------------
;
; convert -- convert a parameter bitMap and rectangle to our internal form.
;
; calling sequence:
;       lea     bitMap,A0 ; point to the bitmap
;       lea     rect,A1         ; and the rectangle inside it
;       lea     ours,A2         ; and our data structure
;       bsr     CONVERT         ; call us
;
; when done, all fields of the "ours" structure are filled in:
;       oBase is address of first byte in which any bits are to be changed
;       oLbits is number of bits into that first byte which are ignored
;       oStride is the stride from one row to the next, in bits
;       oCols is the number of columns in the rectangle
;       oRows is the number of rows
;
; registers used: D0, D1, D2
;

CONVERT

;
; save the starting word and bit address of the stuff:
;
        move.w  top(A1),D0              ; pick up top of inner rectangle
        sub.w   bounds+top(A0),D0       ; figure rows to skip within bitmap
        mulu    rowbytes(A0),D0         ; compute bytes to skip (relative offset)

        add.l   baseaddr(A0),D0         ; find absolute address of first row to use

        move.w  left(A1),D1             ; pick up left coordinate of inner rect
        sub.w   bounds+left(A0),D1      ; find columns to skip
        move.w  D1,D2                   ; copy that
        and.w   #7,D2                   ; compute bits to skip in first byte
        move.w  D2,oLbits(A2)           ; save that in the structure

        lsr.w   #3,D1                   ; convert column count from bits to bytes
        ext.l   D1                      ; convert to a long value, so we can...
        add.l   D1,D0                   ; add to row start in bitmap to find 1st byte
        move.l  D0,oBase(A2)            ; save that in the structure

;
; save stride of bitmap; this is same as for the original, but in bits.
;
        move.w  rowbytes(A0),D0         ; pick up the stride
        lsl.w   #3,D0                   ; multiply by eight to get a bit stride
        move.w  D0,oStride(A2)          ; stick it in the target structure

;
; save the number of rows and columns.
;
        move.w  bottom(A1),D0           ; get the bottom of the rectangle
        sub.w   top(A1),D0              ; less the top coordinate
        sub.w   #1,D0                   ; get number of highest row (1st is zero)
        bmi.s   CERROR                  ; nothing to do?  (note: 0 IS ok)
        move.w  D0,oRows(A2);           ; save that in the structure

        move.w  right(A1),D0            ; get the right edge of the rectangle
        sub.w   left(A1),D0             ; less the left coordinate
        sub.w   #1,D0                   ; make it zero-based
        bmi     CERROR                  ; nothing to do here?
        move.w  D0,oCols(A2)            ; save that in the structure
```

```
;
; all done.  return.
;
        rts

;
; error found in CONVERT.  pop return and jump to the error routine, such as it is.
;
CERROR
        addq.l    #4,SP                ; pop four bytes of return address.
        bra.s     ERROR                ; return silently

;
; ---------------------------------------------------------------------------
;
; log2 -- find the ceiling of the log, base 2, of a number.
; bitwidth -- find how many bits wide a number is
;
; calling sequence:
;       move.l    N,D0                 ; store the number in D0
;       bsr       LOG2                 ; call us
;       move.w    D0,...               ; D0 contains the word result
;
; registers used: D2, (D0)
;

BITWIDTH
        sub.l     #1,D0                ; so 2**n works right (sigh)
LOG2
        tst.l     D0                   ; did they pass us a zero?
        beq.s     LOGDONE              ; if D0 was one, answer is zero
        move.w    #32,D2               ; initialize count
LOG2LP
        lsl.l     #1,D0                ; slide bits to the left by one
        dbcs      D2,LOG2LP ; decrement and loop until a bit falls off

        move.w    D2,D0                ; else save our value where we promised it
LOGDONE                                ; here with final value in D0
        rts                            ; and return

.end    ; procedure dissBits
```

End Listing

Listing One (Text begins on page 120.)

```
Listing 1.  Using the predefined NUMERIC_ERROR Ada exception.


function Power(BASE, EXPONENT : FLOAT) return FLOAT is

begin

   return Exp(Exponent * Ln(Base));

-- This is the area to handle exceptions
exception
   when NUMERIC_ERROR =>
      if Base = 0 then
         return 0;
      else -- return "infinity"
        return FLOAT'FIRST;
      end if;

end Power;
```

End Listing One

Listing Two

```
Listing 2.  General form of exception handling block.


procedure Big_Trouble is

   Negative_Absolute_Temperature,
   Negative_Pressure, Negative_Volume : exception;

   Temperature, Pressure, Volume : FLOAT;

begin

  -- procedure to calculate temperature, Pressure and volume
  -- Calculate temperature in Rankin
  if Temperature < 0.0 then
      raise Negative_Absolute_Temperature;
  end if;

  -- Calculate pressure and volume
  if Pressure < 0.0 then
      raise Negative_Pressure;
  end if;

  if Volume < 0.0 then
      raise Negative_Volume;
  end if;

  -- other procedure statements

 exception -- handling block
   when NUMERIC_ERROR =>

        -- handle bad function arguments, underflow or overflow

   when Negative_Absolute_Temperature =>

        -- handle negative absolute temperature results

   when Negative_Pressure | Negative_Volume =>

        -- handle negative pressure or volume values

   when others =>

        -- handle all other problems

 end Big_trouble;
```

End Listing Two

Listing Three

```
Listing 3.  Ada exception handling scope.

procedure The_Boss is
    Boss_Angry : exception;

    procedure Command_Worker is

    begin

       -- sequence of statements
       if income < 0.0 then raise Boss_Angry; end if;
       -- sequence of statements

    end Command_Worker;

    procedure Command_Foreman is

    begin

       -- sequence of statements
       Command_Worker;
       -- sequence of statements
    exception
       when Boss_Angry =>
          -- Try to deal with the boss
    end Command_Foreman;

begin
    -- sequence of statements
    Command_Worker;
    Command_Foreman;
    - sequence of statements
    exception
       when Boss_Angry =>
           -- fire foreman
end The_Boss;
```

End Listing Three

Listing Four

```
Listing 4.  The retry approach with exception handlers.


with TEXT_IO; use TEXT_IO;

procedure Days_of_our_lives;
   type Day_Name is (Sun, Mon, Tue, Wed, Thu, Fir, Sat);
   package DAY_IO is new TEXT_IO.ENUMERATION_IO (Day_Name);
   use Day_IO;

   -- define time-out
   Time_Out : constant integer := 5;
   -- define variable
   Day : Day_Name;
   -- define exception
   Wrong_Day : exception;

begin
   for Count in 1..Time_Out loop

      PUT("What day is it?"); NEW_LINE;

      begin -- exception handling block starts here
        GET(Day); NEW_LINE;
        PUT("Have a nice "); PUT(Day); NEW_LINE;
        exit; -- exit for loop when answer is correct

      exception
         when CONSTRAINT_ERROR =>
           if Count = Time_Out then
              PUT("Sorry! Loop time-out");
              raise Wrong_Day;
           else
              PUT("Sorry! No such weekday"); NEW_LINE;
              PUT("You have "); PUT(Time_Out - Count);
              PUT(" more times to try); NEW_LINE;
              PUT("Let us try once more"); NEW_LINE;
           end if;
      end; -- end error handler
   end loop; -- end for loop
end Days_of_our_lives;
```

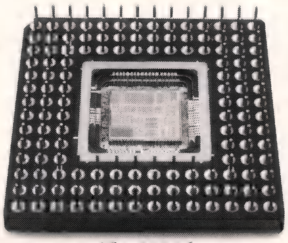
End Listing Four

Listing Five

```
Listing 5.  Using an alternative method with exception handlers.


with TEXT_IO; use TEXT_IO;
procedure Root is

Result, Guess1, Guess2, Accuracy : FLOAT;
Max_Iter : INTEGER;
Diverge, Fatal_Error : exception;


function F(X : FLOAT) return FLOAT is

begin
   return X * X * X - 5.0;
end F;


procedure Newton(Guess, Accuracy : FLOAT; Max_Iter : INTEGER) is
-- Newton's method to find the root of a function
```

(continued on page 113)

A TRAINING COURSE FOR PEOPLE WHO LUST AFTER POWER.



The 80386

Intel is offering three new courses on the world's most powerful 32-bit microprocessor – the 80386. Plus a new intensive course on the 80286.

These in-depth learning sessions are designed for engineers and programmers who want to utilize the full power and potential of these lightning-fast chips.

Lectures are combined with hands-on workshops to provide real-life situations. Allowing you to apply new concepts and techniques immediately.

Courses include:

80386 System Software
80386 Programming using ASM386
High-end Microprocessor Hardware Design
The new 80286 Microprocessor Family Course

Complete training sessions and courses can be scheduled at your facility, or at our training centers.

In addition to training, Intel offers hardware/software support and consultants.

For more complete course information and schedules, call toll-free (800) 548-4725.

Or to register now, contact one of the Intel Training Centers listed below.

Intel Training Centers

Boston Area, Westford Corp. Ctr.
Three Carlisle Road, 1st Floor
Westford, MA 01886
(617) 692-1000

Chicago Area
300 N. Martingale Road, Suite 300
Schaumburg, IL 60194
(312) 310-5700

San Francisco Area
2700 San Tomas Expressway
Santa Clara, CA 95051
(408) 970-1700

Washington, D.C. Area
7833 Walker Drive, 5th Floor
Greenbelt, MD 20770
(301) 220-3380

intel®

DR. DOBB'S CATALOG

featuring:

DR. DOBB'S TOOLBOOK OF 68000 PROGRAMMING

A complete collection of practical programming tips and techniques for the 68000 family, including the best articles from Dr. Dobb's along with much new material!

STAT TOOLBOX FOR TURBO PASCAL

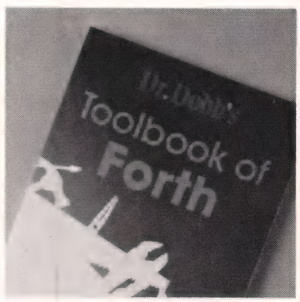
Two statistical packages in one! A reference disk and manual show you the tools you need to build your own statistical applications, while a demonstration disk and manual provide ready-to-use, fully functioning statistical programs.

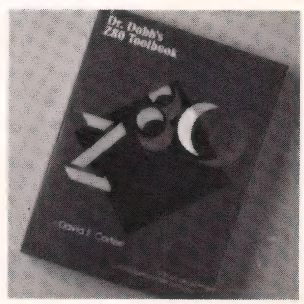
TAMING DOS

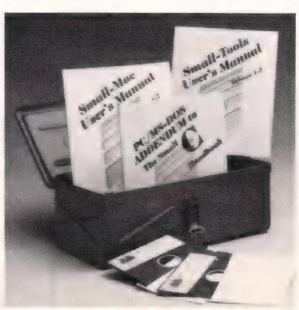
Learn to make DOS work for YOU! Taming DOS takes you beyond the basics so you can customize DOS to fit YOUR needs.

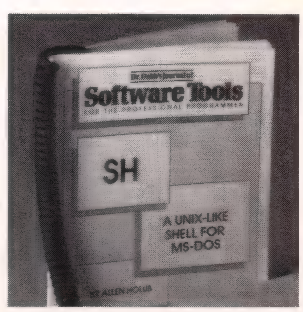
ALSO INSIDE:

A UNIX-LIKE SHELL &
UTILITY PACKAGE FOR MS-DOS
DR. DOBB'S COMPLETE C TOOLBOX
THE TOOLBOOK OF FORTH
Z80 TOOLBOOK









M&T BOOKS

SH: A UNIX-LIKE SHELL FOR MS-DOS

Includes complete source code and documentation

THE SHELL V.2
NEW, UPGRADED VERSION!

The shell is an MS-DOS implementation of the most often used parts of the UNIX C Shell. This package includes an executable version of the Shell, along with **complete source code and full documentation**, by DDJ columnist Allen Holub. If you are a registered user, or have already purchased Version 1 of the Shell from Dr. Dobb's, you can receive the upgrade disk for only $6. Supported features include:

Editing Command-line editing with the cursors is supported. The line is visible as you edit it.
Aliases Can be used to change the names of commands or as very fast, memory-resident, batch files. Nested aliases are supported.
History You can execute previous commands. The command can be edited before being executed. Version 2 supports imbedded history requests (Bar; !! >foo).
Redirection and Pipes <> >> >& >>& ¦
Pipe temporary files can be put on a RAM disk.
Unix-like Command Syntax / can be used to separate directory names (\ can now be used as well). A 2048-byte commnd line is supported. Command-line wild card expansion. Multiple commands on a line.
DOS-compatible prompt support
$d $t $_ $e $h $n $q $$ $%
C-Shell Based Shell Scripts (batch files) Shell Variables are macros that can be used on the command line. Version 2 supports arithmetic manipulation of shell variables using the @ command. The following C operators are supported: () + − * / % <= >= <> != == ! && ¦¦ =
A batch file can call another batch file like a subroutine. Control is passed to the second file and then back to the first when the second is finished. Batch files can return values to the calling file using the exit and $status mechanisms.

A powerful, interpretive, programming language, based on the UNIX C Shell, is now supported, including:

if/then/else	foreach	break
while	switch/case	continue

All commands can be nested.
The shell runs on IBM PC's and compatibles.

The Shell	**Item #160**	**$29.95**
The Shell Upgrade Disk *(for owners of the Shell Version 1)*	**Item #160A**	**$6.00**

SAVE OVER 15%! ORDER THE SHELL AND /UTIL TOGETHER FOR ONLY $50!

Shell /Util Package	**Item #162**	**$50**

/UTIL

/Util is a collection of UNIX-like utility programs for MS-DOS. This package includes updates of the highly acclaimed Dr. Dobb's articles; Grep: a UNIX-like Generalized Regular Expression Processor, and LS and Getargs from DDJ's C Chest.

Source code is included and all programs (and most of the utility subroutines) are fully documented in a UNIX-style manual. You'll find executable versions of:

cat	echo	mv	rm
cp	grep	p	rmdir
date	Ls	pause	sub
du	mkdir	printenv	chmod

/Util	**Item #161**	**$29.95**

TO ORDER: RETURN THE FORM AT THE END OF THE CATALOG, OR CALL TOLL-FREE 1-800-528-6050 EXT 4001 AND REFER TO PRODUCT ITEM NUMBER, TITLE AND DISK FORMAT

THE DR. DOBB'S TOOLBOOK SHELF

Z80 TOOLBOOK

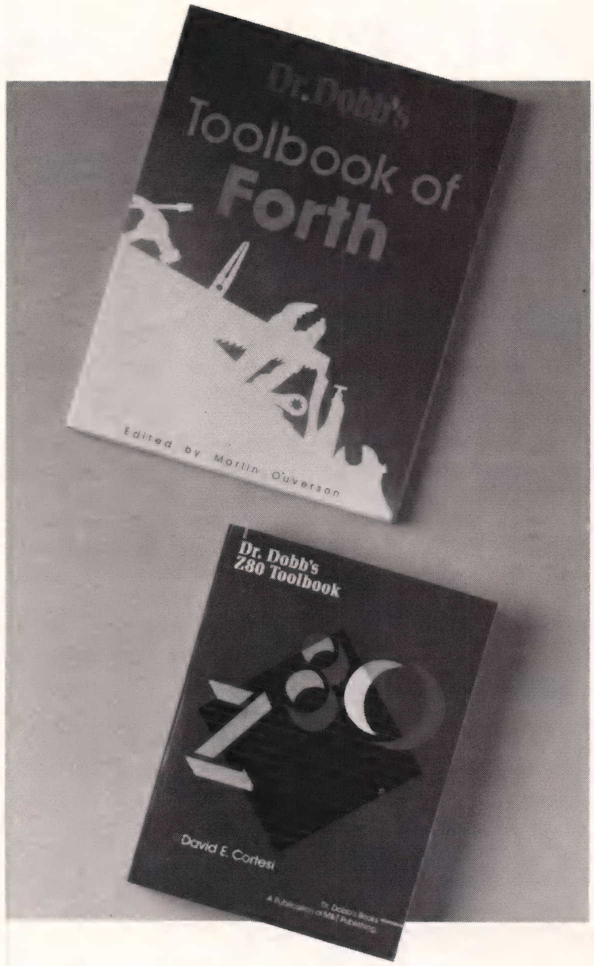
David E. Cortesi, longtime Dr. Dobb's columnist brings you:

- **a method of designing programs and coding them in assembly language.** Cortesi walks through the initial specifications, designing an algorithm and writing the code. He demonstrates the construction of several useful programs.
- **a complete, integrated toolkit of subroutines** for arithmetic, for string-handling, and for total control of the CP/M file system. They bring the ease and power of a compiler's run-time library to your assembly language work, without a compiler's size and sluggish code.
- **Every line of the toolkit's source code is there to read.**

ORDER THE Z80 SOFTWARE ON DISK!

All the software in Dr. Dobb's Z80 Toolbook—the programs plus the entire toolkit, both as source code and object modules for both CP/M 2.2 and CP/M Plus—is yours on disk! Most of the programs are included in the book, however, the disk is necessary for complete listings. A Z80 microprocessor and a Digital Research International RMAC assembler or equivalent are required.

Dr. Dobb's Z80 Toolbook **Item #022** **$25**
Dr. Dobb's Z80 Toolbook w/disk **Item #022A** **$40**
Please specify one of the following disk formats: 8" SS/SD, Apple, Osborne, or Kaypro



DR. DOBB'S TOOLBOOK OF FORTH

This comprehensive collection of useful Forth programs and tutorials contains DDJ's best Forth articles, expanded and revised along with new material. In addition, you'll glean important insights about the potential of this increasingly popular language from the many in-depth discussions of advanced Forth topics. You'll find sections on:

Mathematics in Forth, including "Series Expansion in Forth," "Forth Floating-Point Package," and "Signed Integer Division"
Modifications/Extensions, including "A Proposal for Strings in Forth," "Non-Deterministic Control Words," "Some Forth Coding Standards," and "Towards a More Writable Forth Syntax"
Forth Programs, including "GO in Forth," "Elements of a Forth Data-Base Design," "The Forth Sort," "SEND & RECV," "Interface for a Mouse," "Relocating Loader in Forth," "Forth Decompiler," "Screen-Oriented Editor ReVisited," "Evolution of a Video Editor," "H-19 Screen Editor," and "The Conference Tree."
Forth—the language, including "The Forth Philosophy," "Teaching Forth as a First Language," and "Forth-83 and Vocabularies"
Implementing Forth, including "Forth and the Motorola 68000," "A 68000 Forth Assembler," "A Forth Assembler for the 6502," and "Z8000 Forth." You'll also find Appendices that will help you convert fig-Forth to Forth-83, and tell you how to stay up-to-date on the latest developments and refinements of this popular language.

The screens in the book are also available on disk as ASCII files. Receive Dr. Dobb's Toolbook of Forth, along with the software on disk, together for only $39.95.

Dr. Dobb's Toolbook of Forth **Item #030** **$22.95**
Dr. Dobb's Toolbook of Forth w/Disk **Item #031** **$39.95**
Please specify MS/PC-DOS, Apple II, Macintosh, or CP/M. For CP/M disks, specify Osborne or 8" SS/SD.

TO ORDER: RETURN THE FORM AT THE END OF THE CATALOG, OR CALL TOLL-FREE 1-800-528-6050 EXT 4001 AND REFER TO PRODUCT ITEM NUMBER, TITLE AND DISK FORMAT

DR. DOBB'S C TOOLBOX: POWERFUL

DR. DOBB'S C SOFTWARE TOOLS ON DISK

To complement the *Toolbook, Dr. Dobb's* offers the following programs on disk. Documentation and full C source code are included.
Except where indicated, both CP/M and MS/PC-DOS versions are available.

SMALL-C COMPILER

Jim Hendrix's Small-C Compiler is the most popular piece of software published in *Dr. Dobb's* 11-year history. Like a home-study course in compiler design, the Small-C Compiler and the *Small-C Handbook* provide everything you need but the computer for learning how compilers are constructed, and for learning C at its most fundamental level.

Available for MS/PC-DOS or CP/M systems. Please specify format.

Small-C Compiler **Item #007** **$19.95**

SMALL-MAC:
AN ASSEMBLER FOR SMALL-C

This assembler features simplicity, portability, adaptability, and educational value. The package includes:

- a simplified macro facility
- C language expression operators
- object file visibility
- descriptive error messages
- an externally defined instruction table

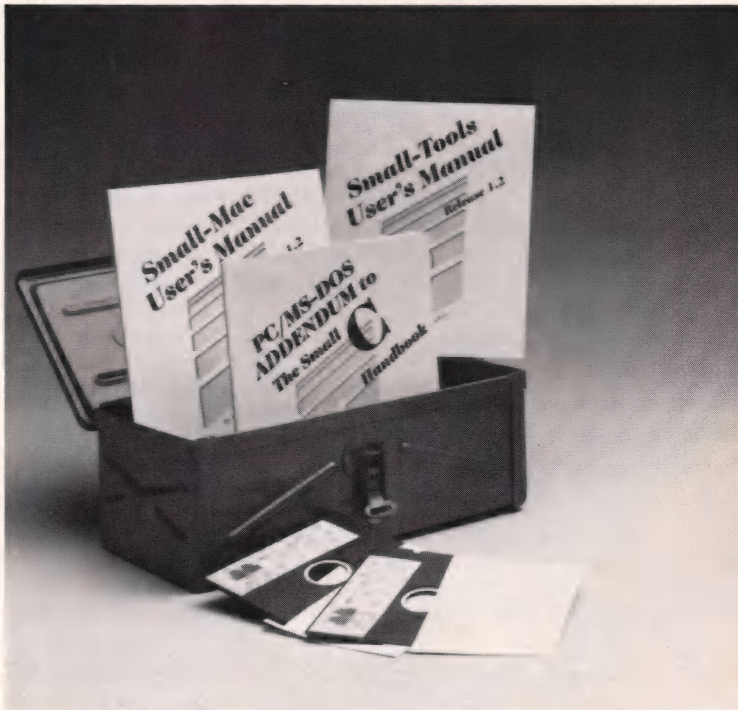
You get the macro assembler, linkage editor, load-and-go loader, library manager, CPU configuration utility, and a utility to dump relocatable files. Documentation is also included.
For CP/M systems only. Please specify format.

Small-Mac **Item #012A** **$29.95**

SMALL-TOOLS:
PROGRAMS FOR TEXT PROCESSING

This package of programs performs specific, modular operations on text files, including: editing; formatting; sorting; merging; listing; printing; searching; changing; transliterating; copying; concatenating; encrypting and decrypting; replacing spaces with tabs and tabs with spaces; counting characters, words, or lines; and selecting printer fonts.
Small-Tools is supplied in source code form so you can select and adapt these tools to your own purposes. Documentation is also included.

Small-Tools **Item #010A** **$29.95**



DR. DOBB'S TOOLBOOK OF C

Over 700 pages of C material, including articles by such C experts as Kernighan and Ritchie, Cain and Hendrix, Skjellum and Holub! The level is sophisticated and pragmatic, appropriate for professional C programmers.
The most valuable part of the ***Toolbook*** to many will be the hundreds of pages of useful C source code, including:

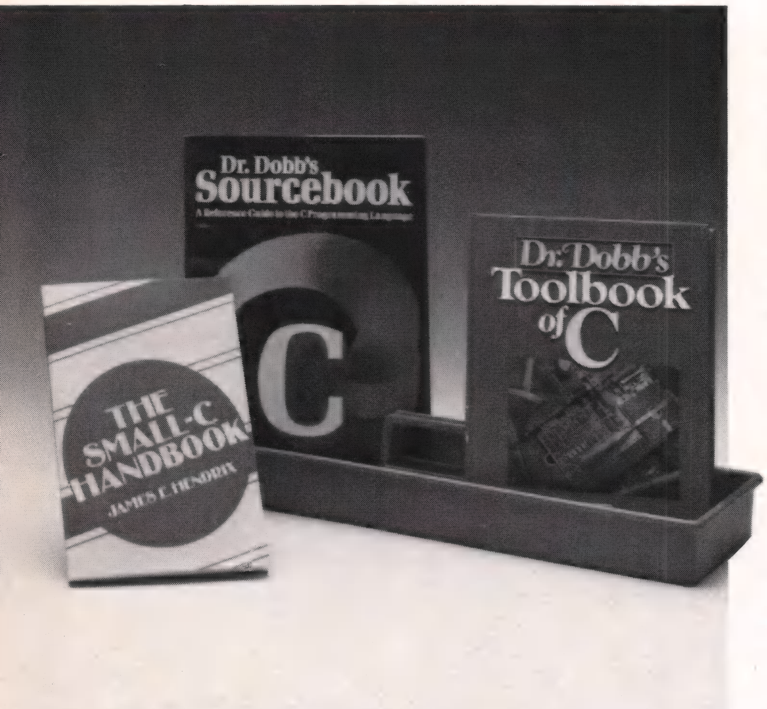
- Jim Hendrix's famous Small-C Compiler and New Library for Small C—Also available on disk!
- NEW! Hendrix's Small Mac: An Assembler for Small C and Small Tools: Programs for Text Processing—Both also available on disk!
- All of Anthony Skjellum's C Programmer's Notebook columns distilled by Tony into one thought-provoking chapter

The accompanying text explains, in the programmers own words, why they did what they did.
You'll find all the best C articles and code published in ***Dr. Dobb's*** over the years updated for the ***Toolbook***, including Ron Cain's original Small-C article.
From M&T Publishing and Brady Communications

Dr. Dobb's Toolbook of C **Item #005** **$29.95**

TO ORDER: RETURN THE FORM AT THE END OF THE CATALOG, OR
CALL TOLL-FREE 1-800-528-6050 EXT 4001
AND REFER TO PRODUCT ITEM NUMBER, TITLE AND DISK FORMAT

TOOLS FOR SOFTWARE DEVELOPERS



DR. DOBB'S SOURCEBOOK:

THE C PROGRAMMING LANGUAGE REFERENCE

A comprehensive reference manual to new information, products and services, the Sourcebook contains:

- a bibliography of over 300 articles and books on C
- a descriptive list of products for C programmers, including compilers, editors, interpreters, and utilities
- a list of C-related services: classes, seminars, and on-line services

Dr. Dobb's Sourcebook **Item #004** **$7.95**

SPECIAL PACKAGES 20% OFF

Receive a complete set of Dr. Dobb's C programming tools for your MS/PC-DOS or CP/M system for 20 percent off the combined individual prices!

CP/M C PACKAGE

Receive this special package and save $20! You'll get:

- Dr. Dobb's Toolbook for C
- The Small-C Handbook
- The Small-C Compiler on disk
- The Small-Mac assembler on disk, with documentation
- The Small-Tools text-processing programs on disk, with documentation all for only $99.95!

Please specify format.

CP/M C Package **Item #005A** **$99.95**

MS/PC-DOS C PACKAGE

Save $20 when you order this special package. You'll receive:

- Dr. Dobb's Toolbook of C
- The Small-C Handbook with the MS/PC DOS Addendum
- The Small-C Compiler on disk
- The Small Tools text-processing programs on disk, with manual all for only $82.95

MS/PC-DOS C Package **Item #005B** **$82.95**

THE SMALL-C HANDBOOK

Jim Hendrix's *Small-C Handbook* is the reference book on his Small-C Compiler. In addition to describing the operation of the compiler, the book contains complete source listings to the compiler and its library of arithmetic and logical routines.
A perfect companion to the Hendrix Small-C Compiler available from *DDJ* on disk, the *Handbook* even tells you how to use the compiler to generate a new version of itself!
While both the *Handbook* and the *Toolbook* provide documentation for the Small-C Compiler, the *Handbook* contains a more detailed discussion and is available with addendum for the MS/PC-DOS version.
From M&T Publishing and Brady Communications

The Small-C Handbook **Item #006** **$17.95**

The Handbook with MS/PC-DOS Addendum **Item #006A** **$22.95**

C DISK FORMATS

When ordering, please indicate MS/PC DOS or CP/M. For CP/M disks, please specify one of the follwing formats: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8" SS/SD. Special order formats are available for an additional $10 each.

TO ORDER: RETURN THE FORM AT THE END OF THE CATALOG, OR CALL TOLL-FREE 1-800-528-6050 EXT 4001 AND REFER TO PRODUCT ITEM NUMBER, TITLE AND DISK FORMAT

DR. DOBB'S LATEST RELEASES

DR. DOBB'S TOOLBOOK OF 68000 PROGRAMMING

In this complete collection of practical programming tips and techniques for the 68000 family, you'll find the best articles on 68000 programming ever published in Dr. Dobb's, along with new material from 68000 experts. You'll learn about the most important features of the 68000 microprocessor from a full description and concise discussion of its history and design. And, useful applications and examples will show you why computers using the 68000 family are easy to design, produce and upgrade. **Contents include:**

an Introduction to the 68000 Family
- 68000 Instruction Set

Development Tools
- Bringing Up the 68000: A First Step
- A 68000 Cross-Assembler

Useful 68000 Routines and Techniques
- A Simple Multitasking Kernel for Real-Time Applications
- The Worm Memory Test
- A Mandelbrot Program for the Macintosh

All programs are available on disk!

In addition, an executable version of the 68000 Cross-Assembler can be purchased along with the source code and documentation for $25. (Requires one or more disk drives and either CP/M-80, CP/M 2.2 with 64 bytes or MS-DOS with 128 bytes)

Dr. Dobb's 68000 Programming Toolbook	**Item #040**	**$29.95**
68000 Toolbook with Disk	**Item #041**	**$49.95**

Please specify one of the following disk formats: CP/M 8″, Osborne, Macintosh, Amiga, Atari 520st, MS-DOS

68000 Cross-Assembler	**Item #042**	**$25**

STAT TOOLBOX FOR TURBO PASCAL

The STAT TOOLBOX is two statistical packages in one! Whether you're a programmer looking for tools to build your own applications, or a user who wants a complete, fully functioning package, the STAT TOOLBOX will bring convenience, power and versatility to your statistics programs!
The STAT TOOLBOX is written in Turbo Pascal and includes full source code. It contains two complete packages:

A reference disk and manual
Flexible, time-saving building blocks allow you to customize statistical applications to fit your needs. You'll find:
- statistical distribution functions;
- random number generation;
- basic descriptive statistics
- parametric and non-parametric statistical testing.
- bivariate linear regression, multiple and polynomial regression
- automatic best curve selection

A demonstration disk and manual
The demonstration package includes fully functioning statistical programs, and two data management programs. In addition to these practical tools, files containing sample data will help users learn to enter, edit, maintain and manipulate data.
High resolution graphics capabilites include drawing histograms and regression lines, and plotting residuals for regression analysis. (Borland's Turbo Graphic's Toolbox is required)
For IBM PC's and compatibles. Turbo Pascal version 2.0 or later, and PC-DOS 2.0 or later are required.

Stat Toolbox	**Item #050**	**$49.95**

TAMING DOS BY THOM HOGAN

Learn how to make DOS work for YOU! **Taming DOS** will take you beyond the basics, picking up where your DOS manual leaves off. You'll find batch files and DOS enhancements that extend the power of DOS so you can work more accurately and efficiently. And, you'll learn how to customize DOS to fit YOUR needs, saving you time and frustration every time you use it.
Taming DOS will show you how to create a memory-resident clock, rename subdirectories and change file attributes. You'll learn more about what's on your disk and how to protect it easily, how to create configurable AUTOEXEC.BAT files and understand redirection and piping. **You'll find:**
- how to customize CONFIG.SYS and use ANSI.SYS to change the appearance of DOS
- extensive batch file coverage
- example routines using redirection, including redirection operators, filters, and pipes
- DOS enhancement programs that are ready to use, with executable commands.
- Full source code, allowing you to alter programs and create a custom system.

The book includes assembly language programs that allow you to manipulate DOS at its lowest form. An assembler is not needed to enter the programs; **Taming DOS** presents an alternative method for anyone with BASIC on their machine. The programs, including batch files and DOS enhancements are also available on disk along with source code.

Taming DOS	**Item #060**	**$19.95**
Taming DOS with disk	**Item #061**	**$34.95**

TO ORDER: RETURN THE FORM AT THE END OF THE CATALOG, OR CALL TOLL-FREE 1-800-528-6050 EXT 4001 AND REFER TO PRODUCT ITEM NUMBER, TITLE AND DISK FORMAT

ORDER FORM

ORDER NOW!

NAME ______________________________

(Please use street address, not P.O. Box)

ADDRESS ______________________________

CITY ______________ STATE ______ ZIP ______

DAY PHONE ______________________________

For disk orders, please indicate format. Refer to ad for standard format availability for each product. Special formats available for additional $10 each.

☐ MS/DOS ☐ CP/M ____ Zenith Z-100 DS/DD
☐ Macintosh ____ Kaypro ____ Osborne ____ Amiga
☐ Apple II ____ 8" SS/SD ____ Apple ____ Atari 520st

For Faster Service
On Credit Card Orders

CALL TOLL FREE
1-800-528-6050
Ext. 4001

Please Refer To Item #
When Ordering

QUANTITY	ITEM #	DESCRIPTION	UNIT PRICE	TOTAL PRICE

CA residents must add applicable sale tax on merchandise total ________ %
(CA residents must add sales tax EXCEPT *Dr. Dobb's Sourcebook* #004)
Shipping must be included with order. See rates below.

SUB-TOTAL ______
SALES TAX ______
SHIPPING ______
TOTAL ORDER ______

☐ VISA — NAME ON CARD ______________________________

☐ MASTERCARD — ACCOUNT NO. ______________________________

☐ AMERICAN EXPRESS — EXPIRATION DATE ______________________________

☐ CHECK (make checks payable to M&T Publishing) — SIGNATURE ______________________________

In U.S. For Bound Volumes, add $3.25 per book. Add $8.75 for special C Packages. For other books and disks, add $2.25 per item.
Outside U.S. For Bound Volumes, add $6.25 per book surface mail. Add $18 surface mail for Special C Packages. For other books and disks, add $5.25 per item surface mail. Foreign airmail rates available on request.

For Faster Service and reduced shipping costs, Europeans may order direct from: Markt & Technik, Buchverlag, Hans-Pinsel-Strasse 2, 8013 Haar bei München. Call Germany 89-4613-221 for prices in Deutch Marks.

PROMPT DELIVERY! DEALER INQUIRY WELCOME!

DR. DOBB'S CATALOG ORDER FORM

Please fold along fold-line and staple or tape closed.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

First Class Permit No. 790 Redwood City, CA

Postage Will Be Paid By Addressee

DR. DOBB'S CATALOG

501 GALVESTON DRIVE
REDWOOD CITY, CA 94063

Please fold along fold-line and staple or tape closed.

Listing Five (Listing continued, text begins on page 120.)

```
Funct, Derivative, h, Diff : FLOAT;

begin
   loop
      if ABS(Guess) > 1.0 then h := 0.01 * Guess;
                          else h := 0.01;
      end if;

      Funct := F(Guess);
      Derivative := (F(Guess + h) - Funct) / h;
      Diff := Funct / Derivative;
      Guess := Guess - Diff;

      Max_Iter := Max_Iter - 1;
      if Max_Iter < 0 then
         raise Diverge;
      end if;


      if ABS(Diff) <= Accuracy then exit; end if;
   end loop;

   PUT(Guess);

end Newton;


procedure Bisection(A, B, Accuracy : FLOAT; Max_Iter : INTEGER) is
-- Bisection method to find the root of a function

Mean, FA, FB, FM : FLOAT;

begin
   FA := F(A); FB := F(B);
   -- Get midpoint estimate for the root
   Mean := (A + B) / 2.0;

   while ABS(A - B) > Accuracy loop

     FM := F(Mean);
     -- Does A and Mean have same function sign?
     if FM * FA > 0.0
     then
        A := Mean; FA := FM;
     else
        B := Mean; FB := FM;
     end if;

     -- Get midpoint estimate for the root
     Mean := (A + B) / 2.0;

     Max_Iter := Max_Iter - 1;
     if Max_Iter < 0 then
         raise Fatal_Error;
     end if;

   end loop;

   PUT(Mean);
end Bisection;

begin -- Root --
  PUT("Enter first guess for the root "); GET(Guess1); NEW_LINE;
  PUT("Enter second guess for the root "); GET(Guess2); NEW_LINE;
  PUT("Enter desired accuracy"); GET(Accuracy); NEW_LINE;
  PUT("Enter maximum number of iterations "); GET(Max_Iter);
  NEW_LINE; NEW_LINE;
  PUT("Root = ");
  begin -- start outer exception handler
    -- Try Newton's method first
    Newton(Guess1, Accuracy, Max_Iter);
    exit; -- terminate program successfully
  exception
    when NUMERIC_ERROR | Diverge =>
       begin -- start inner exception handler
         -- This method will definitely work, but is slower
         Bisection(Guess1, Guess2, Accuracy);
         exit; -- terminate successfully with second method
       exception
         when others =>
            PUT("Fatal Error.  Cannot recover");
            NEW_LINE;
       end; -- inner exception handler
  end; -- outer exception handler
end Root;
```

End Listing Five

Listing Six

```
Listing 6.  The clean up method used in exception handlers.


with TEXT_IO; use TEXT_IO;
procedure Root is

Result, Guess, Accuracy : FLOAT;
Max_Iter : INTEGER)
Diverge : exception;


function F(X : FLOAT) return FLOAT is

begin
   return X * X * X - 5.0;
end F;


procedure Newton(Guess, Accuracy : FLOAT; Max_Iter : INTEGER) is
-- Newton's method to find the root of a function

Funct, Derivative, h, Diff : FLOAT;

begin
   loop
      if ABS(Guess) > 1.0 then h := 0.01 * Guess;
                          else h := 0.01;
      end if;
```

(continued on next page)

Listing Six (Listing continued, text begins on page 120.)

```
      Funct := F(Guess);
      Derivative := (F(Guess + h) - Funct) / h;
      Diff := Funct / Derivative;
      Guess := Guess - Diff;

      Max_Iter := Max_Iter - 1;
      if Max_Iter < 0 then
          raise Diverge;
      end if;


      if ABS(Diff) <= Accuracy then exit; end if;
   end loop;

   NEW_LINE; NEW_LINE;
   PUT("Root = "); PUT(Guess);
   NEW_LINE; NEW_LINE;
end Newton;

begin -- Root --
  PUT("Enter guess for the root "); GET(Guess); NEW_LINE;
  PUT("Enter desired accuracy"); GET(Accuracy); NEW_LINE;
  PUT("Enter maximum number of iterations "); GET(Max_Iter);
  loop

    begin -- start exception handler
      -- Try Newton's method first
      Newton(Guess, Accuracy, Max_Iter);
      exit; -- exit open loop and terminate program successfully
    exception
      when Diverge =>
          PUT("Enter guess for the root ");
          GET(Guess); NEW_LINE;
    end; -- exception handler
  end loop;
end Root;
```

End Listing Six

Listing Seven

```
Listing 7.  Module SafeLib0, a subset of MathLib0 with error
                                                    trapping features.

DEFINITION MODULE SafeLib0;
(* Definition module of SafeLib0, the safer version of MathLib0 *)

(* The EXPORT is not needed for new Modula-2 definition *)
EXPORT QUALIFIED SQRT, LN, EXP, EXPRANGE;

(* Largest argument for exp(X) that yields exp() = 9.9999E+99 *)
CONST EXPRANGE = 230.26;


PROCEDURE SQRT(X : REAL; VAR ArgumentERROR : BOOLEAN) : REAL;
(* Square root function with an argument error flag *)

PROCEDURE LN(X : REAL; VAR ArgumentERROR : BOOLEAN) : REAL;
(* Natural logarithm function with an argument error flag *)


PROCEDURE EXP(X : REAL; VAR ArgumentERROR : BOOLEAN) : REAL;
(* Exponential function with an argument error flag *)


PROCEDURE GetNext(Current, MaxFlag : CARDINAL;
                  VAR Found : BOOLEAN;
                  ErrorFlag : ARRAY OF BOOLEAN) : CARDINAL

END SafeLib0.


IMPLEMENTATION MODULE SafeLib0;

FROM MathLib0 IMPORT sqrt, exp, ln;

PROCEDURE SQRT(X : REAL; VAR ArgumentERROR : BOOLEAN) : REAL;
(* Square root function with an argument error flag *)

BEGIN
   ArgumentERROR := FALSE;

   IF X < 0.0 THEN
       ArgumentERROR := TRUE;
       X := ABS(X)
   END;

   RETURN sqrt(X)

END SQRT;


PROCEDURE LN(X : REAL; VAR ArgumentERROR : BOOLEAN) : REAL;
(* Natural logarithm function with an argument error flag *)
                 ELSE X := 10.0
      END;
   END;

   RETURN ln(X)

END LN;


PROCEDURE EXP(X : REAL; VAR ArgumentERROR : BOOLEAN) : REAL;
(* Exponential function with an argument error flag *)

BEGIN
   ArgumentERROR := FALSE;

   IF X > EXPRANGE
   THEN
       ArgumentERROR := TRUE;

BEGIN
   ArgumentERROR := FALSE;

   IF X <= 0.0 THEN
       ArgumentERROR := TRUE;
       IF X < 0.0 THEN X := ABS(X)

       X := 1.0 / EXPRANGE
   END;

   RETURN exp(X)

END EXP;


PROCEDURE GetNext(Current, MaxFlag : CARDINAL;
                  VAR Found : BOOLEAN;
                  ErrorFlag : ARRAY OF BOOLEAN) : CARDINAL;

VAR Last : CARDINAL;
```

```
BEGIN
   Last := HIGH(ErrorFlag);
   IF MaxFlag > Last THEN MaxFlag := Last END;
   Found := FALSE;
   WHILE (Current <= Last) AND (NOT Found) DO
      IF ErrorFlag[Current] THEN Found := TRUE END;
      INC(Current);
   END;

   RETURN Current

END GetNext;

END SafeLib0.
```

End Listing Seven

Listing Eight

```
Listing 8.  Module SafeLib1, a second alternate subset of
MathLib0 with error trapping features.


DEFINITION MODULE SafeLib1;
(* Definition module of SafeLib1, the safer version of MathLib1 *)

(* The EXPORT is not needed for new Modula-2 definition *)
EXPORT QUALIFIED SQRT, LN, EXP, EXPRANGE,
                 MAXERRORSTACK, ErrorStack;

(* Largest argument for exp(X) that yields exp() = 9.9999E+99 *)
CONST EXPRANGE = 230.26;
      MAXERRORSTACK = 50;


VAR ErrorStack : RECORD
                   HeightErrorStack : [0..MAXERRORSTACK];
                   FuncName : ARRAY [1..MAXERRORSTACK] OF
                                   ARRAY [0..3] OF CHAR
               END;


PROCEDURE SQRT(X : REAL) : REAL;
(* Square root function *)

PROCEDURE LN(X : REAL) : REAL;
(* Natural logarithm function *)

PROCEDURE EXP(X : REAL) : REAL;
(* Exponential function *)
END SafeLib1.



IMPLEMENTATION MODULE SafeLib1;

FROM MathLib0 IMPORT sqrt, exp, ln;

PROCEDURE SQRT(X : REAL) : REAL;
(* Square root function *)

BEGIN
   IF X < 0. THEN
      PushErrorStack("SQRT");
      X := ABS(X);
   END;

   RETURN sqrt(X)


END SQRT;


PROCEDURE LN(X : REAL) : REAL;
(* Natural logarithm function *)

BEGIN

   IF X <= 0.0 THEN
      ArgumentERROR := TRUE;
      IF X < 0.0 THEN X := ABS(X)
                 ELSE X := 10.0
      END;
   END;

   RETURN ln(X)

END LN;


PROCEDURE EXP(X : REAL) : REAL;
(* Exponential function  *)
```

(continued on next page)

Listing Eight (Listing continued, text begins on page 120.)

```
BEGIN

   IF X > EXPRANGE
   THEN
       ArgumentERROR := TRUE;
       X := 1.0 / EXPRANGE
   END;

   RETURN exp(X)

END EXP;


PROCEDURE ClearErrorStack;

BEGIN
   ErrorStack.HeightErrorStack := 0
END ClearErrorStack;


PROCEDURE PushErrorStack(Name : ARRAY OF CHAR);

VAR I : CARDINAL;

BEGIN
   WITH ErrorStack DO
       INC(HeightErrorStack);

       I := 0;
       WHILE (I <= HIGH(Name)) AND (Name[I] <> 0C) DO
          FuncName[HeightErrorStack,I] := Name[I]
       END;

       IF I < HIGH(Name) THEN FuncName[I+1] := 0C END;

  END; (* WITH *)

 END PushErrorStack;


 PROCEDURE InError() : BOOLEAN;

 BEGIN
   RETURN (ErrorStack.HeightErrorStack > 0)
 END InError;

 BEGIN (* Module initialization *)
   ClearErrorStack
 END SafeLib1.
```

End Listing Eight

Listing Nine

```
Listing 9.  Turbo  Pascal matrix inversion program  using  Turbo
                                                    Extender utilities.

PROGRAM INVERT;

(* Program to test speed of floating point matrix inversion.  *)
(* The program will form a matrix with ones' in every member  *)
(* except the diagonals which will have values of 2.          *)

CONST MAX = 140;
      RArowsPerPage = 20;
      RAcolsPerPage = 20;
      RApagesDown   = 7;
      RApagesAcross = 7;


TYPE RAelementType = REAL;
(*$I RARRAY.INC*)

VAR J, K, L : INTEGER;
    DET, PIVOT, TEMPO : REAL;
    A : RAarrayPtr;
    CH : CHAR;

PROCEDURE SHOW_MATRIX;

BEGIN
    FOR J := 1 TO MAX DO BEGIN
        FOR K := 1 TO MAX DO BEGIN
            WRITE(getRA(A,K,J));
            WRITE('  ');
        END;
        WRITELN;
    END;
END;


BEGIN
    setupRa; (* SETUP BIGARRAY *)
    makeRA(A, 1.0, noinit);

    (* Creating test matrix *)
    FOR J := 1 TO MAX DO BEGIN
        FOR K := 1 TO MAX DO
            setRA(A, K, J, 1.0);
        setRA(A, J, J, 2.0)
    END;

    (* The test below will ensure that the user does not spend   *)
    (* a lot of time looking at a rather obvious matrix when its *)
    (* size is large.                                            *)

    IF MAX <= 10 THEN BEGIN
        WRITELN('Matrix is ');
        SHOW_MATRIX;
        WRITELN; WRITELN;
    END;

    WRITELN('Starting matrix invertion');
    DET := 1.0;
    FOR J := 1 TO MAX DO BEGIN
        PIVOT := getRA(A,J,J);
        DET := DET * PIVOT;
        setRA(A,J,J,1.0);
        FOR K := 1 TO MAX DO
            setRA(A,J,K, (getRA(A,J,K) / PIVOT));

        FOR K := 1 TO MAX DO
            IF K <> J THEN BEGIN
                TEMPO := getRA(A,K,J);
                setRA(A,K,J,0.0);
                FOR L := 1 TO MAX DO
                    setRA(A,K,L, (getRA(A,K,L) - getRA(A,J,L) * TEMPO));

            END;
    END; (* End of outer for-loop *)
    WRITELN('PRESS <CR> to view matrix '); READLN(CH); WRITELN;
    WRITELN('The inverse matrix is ');
    SHOW_MATRIX;
    WRITE('Determinant = ');
    WRITE(DET);
    WRITELN; WRITELN;
END.
```

End Listings

THE PROGRAMMER'S SHOP

helps save time, money and cut frustrations. Compare, evaluate, and find products.

RECENT DISCOVERY

Baby 34, 36 RPG II by California Software - complete mini RPG environments for PC include compiler, editor (SEU), OCL processor. Screen gen., sort, data exchange, workstation I/O. Separate products for compatibility w/IBM System /38, /36, /34. PC $1250

AI-Expert System Dev't

Arity System-incorporate with C programs, rule & inheritance MS $ 295
Experteach - Powerful, no limit on memory size. Samples PC $ 399
EXSYS - Improved. Debug, file & external program access. MS $ 339
Insight 2 + - dB2, language. MS $ 879
LPA MicroProlog Intro w/ APES MS $ 149
LPA MicroProlog Prof. w/APES MS $ 595
Others: ESP ($845), Expert Choice ($449)

AI-Lisp

BYSO - Common, MacLISP compatible. 250 + functions, fast PC $ 150
Microsoft MuLisp 85 MS $ 199
PC Scheme LISP - by TI PC $ 95
TLC LISP - clâsses, **compiler.** MS $ 225
TransLISP - learn fast MS $ 85
Others: IQ LISP ($155), UNX LISP ($59), IQC LISP ($269), WALTZLISP ($149)

AI-Prolog

APT - Active Prolog Tutor - build applications interactively PC $ 65
ARITY Standard - full, 4 Meg Interpreter - debug, C, ASM PC $ 350
COMPILER/Interpreter-EXE PC $ 795
With Exp Sys, Screen - KIT PC $1250
LPA MacProlog - Complete incremental compiler and an interpreter MAC $ 295
LPA MicroProlog - intro MS $ 99
LPA MicroProlog Prof. - full memory MS $ 359
Prolog-86 - Learn Fast, Standard, tutorials, samples MS $ 89
Prolog-86 Plus - Develop MS $ 229
TURBO PROLOG by Borland PC $ 69
Others: Prolog-I ($95), Prolog-2 ($859)

AI-Other

METHODS - SMALLTALK has objects, windows, PC $ 69
Personal Consultant by T.I. PC $ 600
Q'NIAL - Combines APL with LISP. Source or binary. PC $ 359
Smalltalk-80 - Xerox version PC $ 995
Smalltalk/V-graphics PC $ 89

FEATURES

ASMLIB - 170 + Microsoft MASM compatible routines for graphics, floats, trig. Full source. PC $ 135

Test Software - capture and replay 1 complete successful test session. Automatically rerun when your code changes with Showcase. PC $ 135

Free Literature Compare Products

Evaluate products. Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any ***"Packet"*** or Addon Packet □ AI □ ADA, Modula □ BASIC □ "C" □ COBOL □ Editors □ FORTH □ FORTRAN □ PASCAL □ UNIX/PC or □ Debuggers, Linkers.

Our Services:

- Programmer's Referral List
- Compare Products
- Help find a Publisher
- *Evaluation Literature FREE*
- BBS - 7 PM to 7 AM 617-826-4086
- Dealers Inquire
- Newsletter
- Rush Order
- Over 700 products
- National Accounts Center

Basic

Basic Development System - for BASICA; Adds Renum, more. PC $ 105
Basic Development Tools by Sterling Castle PC $ 99
Basic Windows by Syscom PC $ 95
BetterBASIC - all RAM, modules Structure. Full BASICA PC $ 165
8087 Math Support PC $ 89
Run-time Module PC $ 229
Better Tools - for Better Basic PC $ 95
CADSAM FILE SYSTEM - full ISAM in MBASIC source. MS $ 75
GoodBas - maintain code PC $ 95
LPI Basic - MS compatible UNIX $1100
Prof. Basic - Interactive, debug PC $ 79
8087 Math Support PC $ 47
QuickBASIC V2.0-New interface PC $ 75
TRUE Basic - ANSI PC $ 119
Run-time Module PC $ 459

Cobol

LPI Cobol - ANSI '74 UNIX $1200
Macintosh COBOL - full MAC $ 459
MBP - Lev. II, native MS $ 885
MicroFocus Professional - full PC Call
Microsoft Cobol Tools - xref, debugger w/source support. Xenix $359 PC $ 219
Microsoft Version II - upgraded. Full Lev. II, native, screens. MS $ 479
Realia - very fast MS $ 839
Ryan McFarland COBOL MS $ 699
COBOL-8X MS $1049

Editors for Programming

BRIEF Programmer's Editor - undo, windows, reconfigure PC Call
EMACS by UniPress - powerful, multifile, windows Source:$929 $ 299
Epsilon - like EMACS, full C-like language for macros. PC $ 169
Lattice Screen Editor - multiwindow, multitasking Amiga $100 MS $ 109
PC/VI - Custom Software PC $ 129
PMATE - power, multitask 80/86 $ 149

C Libraries-Communications

Asynch by Blaise PC $ 149
Multi-Comm - use w/Multi-C PC $ 149
Software Horizons pack 3 PC $ 119

RECENT DISCOVERY

Unishell - Bourne UNIX shell script compiler translates to C and runs faster. Portable, hand optimizable code. Use as C program generator. PC $395

C Language-Compilers

AZTEC C86 - Commercial PC $499
C86 by CI - 8087, reliable MS $299
Datalight C - fast compile, good code, 4 models, Lattice compatible, Lib source. Dev'rs Kit PC $ 85
HOT C - new, intriguing PC $ 85
Lattice C - from Lattice MS $299
Mark Williams - w/debugger MS $399
Microsoft C 4.0- Codeview MS $299
Wizard C - Lattice C compatible, full sys. III, lint, fast. MS $389

C Language-Interpreters

C-terp by Gimpel - full K & R MS $239
INSTANT C - Source debug, Edit to Run-3 seconds, .OBJs MS $389
Interactive C by IMPACC Assoc. Interpreter, editor, source, debug. PC $225
Introducing C-self paced tutorial PC $109
Run/C Professional MS $189
Run/C Lite - improved MS $109

C Libraries-General

Blackstar C Function Library PC $ 79
C Essentials - 200 functions PC $ 85
C Food by Lattice-ask for source MS $109
C Scientific Subroutines - Peerless MS $139
C Tools Plus (1 & 2) PC $149
C Utilities by Essential - Comprehensive screen graphics, strings, source. PC $139
C Worthy Library - Complete, machine independent MS $269
Entelekon C Function Library PC $119
Entelekon Superfonts for C PC $ 45
Greenleaf Functions-portable, ASM $139
PforCe by Phoenix - objects PC $299

C Libraries-Files

FILES: C Index by Trio - full B + Tree, vary length field, multi compiler
/File is object only MS $ 89
/Plus is full source MS $349
CBTREE - Source, no royalties MS $ 99
CTree by Faircom - no royalties MS $339
dbVISTA - full indexing, plus optional record types, pointers, Network.
Object only - MS C, LAT, C86 $159
Source - Single user MS $429
Source - **Multiuser** MS $849
dBASE Tools for C PC $ 79
dbc Isam by Lattice MS $199

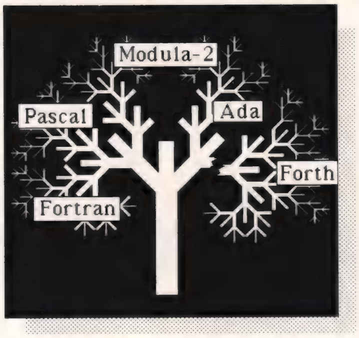
Atari ST & Amiga

We carry full lines of Manx, Lattice, & Metacomco.
Amiga - LINT by Gimpel Amiga $ 79
Cambridge LISP Amiga $200
Lattice C ST, Amiga $139
Lattice Text Utilities Amiga $ 75
Megamax - tight, full ST $200

Circle **no. 141** on reader service card.

We support MSDOS (not just compatibles), PCDOS, Xenix-86, CPM-80, Macintosh, Atari ST, and Amiga.

Error Handling in Ada and Modula-2, Large Turbo Pascal Matrices



In this issue I will discuss error handling (also known as exceptions) in Ada and Modula-2. In the case of Modula-2, I will concentrate on handling errors for several mathematical functions exported by the standard library *MathLib0*. The Pascal section of the column looks at writing programs to handle matrices with sizes greater than 64K using the Turbo Extender package.

Exceptions in Ada

Ada was designed for numerous applications, including real-time systems. The language's designers chose to tackle error handling effectively rather than pay it lip service as did the designers of standard Pascal and Modula-2. As a result, Ada recognizes an explicit error-handling mechanism, namely, the exception. The basic concept behind exceptions is for a program to detect an error condition and direct the program flow control to resume at the exception handling area. No explicit *GOTO* statements are used, although their effect is still attained. Thus, programmers are relieved from using "defensive" programming methods, as is the case with standard Pascal and Modula-2.

Ada has five predefined exceptions —the *CONSTRAINT_ERROR*, *NUMERIC_ERROR*, *PROGRAM_ERROR*, *STORAGE_ERROR*, and *TASKING_ERROR*. Listing One, page 104, shows the predefined *NUMERIC_ERROR* exception

by Namir Clement Shammas

in use with a floating-point power function. In this example, the function body consists of one statement followed by the exception handling block. Ada requires that such blocks be placed at the end of a program or routine (more about the scope of exceptions later). Zero or negative values for the *Base* variable are invalid arguments for the natural logarithm function. The exception handling block examines the value assigned to *Base*. If it equals zero, the function returns a zero; otherwise, it returns the largest negative value that is supported by the implementation. Notice that if the arguments of function *Power* are valid but large in magnitude, an overflow occurs and the largest negative value is also returned.

Ada allows you to define your own exceptions by declaring their names followed by *: exception;*. To invoke user-defined exceptions, use the *raise* keyword followed by the corresponding exception name. Listing Two, page 104, gives a general scheme for defining and raising exceptions. Notice that the first *when* clause in the exception handling block uses the predefined *NUMERIC_ERROR* exception. The second *when* clause tackles negative absolute-temperature values, and the third clause deals with both negative pressures and volumes. The last *when others* clause serves as an *otherwise* catchall error trap. The listing shows how the *if* statement is used in raising user-defined exceptions. Notice how this Ada listing handles erroneous values without resorting to a series of nested *if* statements, as would be the case with Pascal or Modula-2.

Ada enables you to use several exception handlers and control their scope of action and so allows some exceptions to override others. Consider Listing Three, page 104, where procedure *The_Boss* defines the *Boss_Angry* exception and two local procedures, namely *Command_Worker* and *Command_Foreman*. There are two exception handlers for *Boss_Angry*—the first is located inside procedure *Command_Foreman*, the other at the end of the main procedure. When the main procedure starts executing, it first calls for procedure *Command_Worker*. If the *Boss_Angry* exception is raised during the first direct call, the exception handler in the main procedure is invoked. Assuming, on the other hand, that the above call proceeds smoothly, the main procedure resumes normally and invokes procedure *Command_Foreman*. The latter calls for procedure *Command_Worker*. If during this process the *Boss_Angry* exception is raised, the local handler is used instead of the global one.

The role of exception handlers falls into the following categories:

1. Halt program execution.
2. Retry the program.
3. Employ another method or approach.
4. Clean up variables and resume program execution.

Halting the program after displaying an appropriate error message is the least thing a program should do. This course of action uses Ada's exception feature to a minimum because no attempt is made to remedy the error.

The second approach to using exception handlers retries a routine for a fixed number of times to prevent it from being trapped indefinitely. Thus, the program simulates a timeout. Listing Four, page 104, shows a simple example for handling errors associated with enumerated types. The procedure defines a new data type for the weekdays and creates a new set of customized I/O routines. The *Time_Out* constant is defined and set equal to 5, and a user-defined *Wrong_Day* exception is declared.

FORTRAN PROGRAMMERS

Looking for the right PC FORTRAN language system? If you're serious about your FORTRAN programming then you should be using F77L - LAHEY FORTRAN.

"Lahey's F77L FORTRAN is the compiler of choice. It's definitely a 'Programmers FORTRAN,' with features to aid both the casual and the professional programmer... F77L compiled the five files in a total of 12 minutes, which was 4 times as fast as MS FORTRAN and an astounding 6 times as fast as Pro FORTRAN." - PC Magazine

Compare the features and performance of other PC FORTRANs with F77L and you will find that F77L is clearly the superior product.

- Full Fortran 77 Standard (F77L is not a subset)
- Popular Extensions for easy porting of mini and mainframe applications
- COMPLEX*16, LOGICAL*1 and INTEGER*2
- Recursion - allocates local variables on the stack
- IEEE - Standard Floating Point
- Long variable names - 31 characters
- IMPLICIT NONE
- Fast Compile - Increases productivity
- Source On Line Debugger (Advanced features without recompiling)
- Arrays and Commons greater than 64K
- Clear and Precise English Diagnostics
- Compatibility with Popular 3rd Party Software (i.e. Lattice C)
- Easy to use manual
- Technical Support from LCS
- *NEW FEATURE - NAMELIST*

F77L - THE PROGRAMMER'S FORTRAN

$477.00 U.S.

System Requirements: MS-DOS or PC-DOS, 256K, math coprocessor (8087/80287)

FOR MORE INFORMATION: (702) 831-2500

Lahey Computer Systems, Inc.
P.O.Box 6091
Incline Village, NV 89450
U.S.A.

International Dealers:
England: Grey Matter Ltd., Tel: (0364) 53499
Denmark: Ravenholm Computing, Tel: (02) 887249
Australia: Computer Transitions, Tel: (03) 537-2786
Japan: Microsoftware, Inc., Tel: (03) 813-8222

SERVING THE FORTRAN COMMUNITY SINCE 1967

Editor's Choice
- PC Magazine

Circle **no. 186** on reader service card.

C CODE FOR THE PC

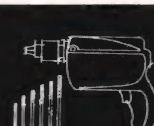
source code, of course

GraphiC 3.0 hi-res color plots	$300
QC88 C Compiler	$90
Concurrent C	$45
Coder's Prolog in C	$45
Translate Rules to C	$30
LEX	$25
YACC & PREP	$25
Small-C compiler for 8088	$20
tiny-c interpreter & shell	$20
Xlisp 1.5a & tiny-Prolog	$20
C Tools	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
(512) 258-0785

Free shipping on prepaid orders — No credit cards

Circle **no. 250** on reader service card.



Power Tools for system builders™

Call today for our free catalog of design aids, compilers, libraries, debuggers, and support tools for Apple and IBM micro computers. The Power Tools catalog includes product descriptions, warranty and license terms, and all the information you need to make an intelligent purchase decision.

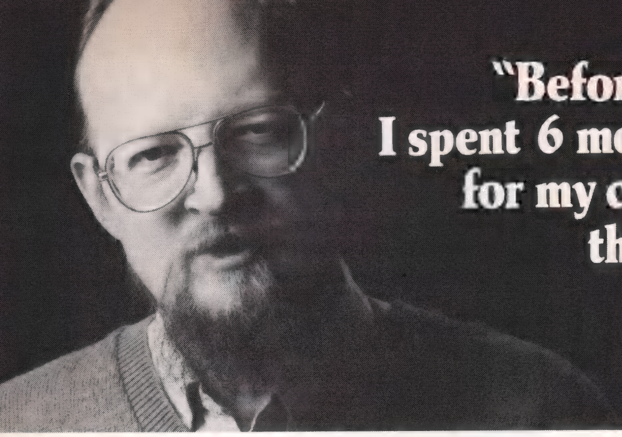
TSF offers technical support, competitive pricing, free UPS shipping on orders over $100, and a reasonable return policy. Visa, MasterCard, and American Express accepted without surcharge. **TSF helps you get your job done.**

Sample Prices:
MS Quick Basic $49.00
Periscope II $129.00
DS Backup $64.95
10 Diskette Mailers $4.95
Gimpel PC Lint $139.00
PCMacBasic $100.00

Call Toll Free
24 hrs a day/7 days a week
Ask For Operator 2053
800-543-6277
Calif: 800-368-7600

TSF
The Software Family™
• Dept. C-2 • 649 Mission Street
• San Francisco • CA 94105
• (415) 957-0111

Circle **no. 230** on reader service card.



"Before I chose Microsoft C, I spent 6 months evaluating C compilers for my company. Now you can do the same in 2 hours."

Bill Davidsen, Software Engineer
Office Automation Group
[redacted]

"Call us. You can get Microsoft C or our comprehensive report on C by the day after tomorrow."

Bruce Lynch, President
The Programmer's Shop

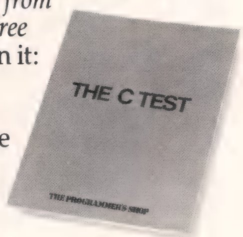
The security of thorough research. It took Bill Davidsen six months to thoroughly evaluate all C products before he selected Microsoft C. For him, its tight code and UNIX System V™ compatibility were exactly what he needed. And now Version 4.00 includes CodeView,™ a source-level windowing debugger.

Thanks to expert users like Bill, and The Programmer's Shop, you can enjoy that satisfied feeling of thorough product evaluation in just a few hours.

We recommend evaluating software by also getting detailed information from several different sources, including unbiased reports and reviews. Bill agrees completely.

In fact, he helped us compile the objective opinions of 4 magazines, 14 users and 3 industry analysts in a 16-page report on C: *The C Test*. It can help you be absolutely sure of making the choice that's best for you. And it's absolutely free.

C for yourself. As an objective evaluation by users and professionals alike, *The C Test* is one of the most comprehensive and informative reports currently available on C development tools. *It's only available from The Programmer's Shop. And it's yours free for the asking.* Here's what you'll find in it:



The C Test ▪ Detailed Tech Specs ▪ Benchmark Source Code ▪ Magazine Reviews ▪ Users' Feedback ▪ Performance Benchmarks ▪ User Study and Profiles ▪ Test Drive Survey Results ▪ 37 Compatible Products

And if you're looking for even more C support, Microsoft-compatible libraries for file management, graphics, screen control, object-oriented programming and other tools are ready to ship.

The best programs for less. We think the only way to serve you is to give you the best programming alternatives. The best recommendations for your needs. To deliver immediately. And this is how we do it.

We start by giving you a choice of over 62 programming language implementations and 174 support programs. All from the same source. All competitively priced.

Our informed programmers offer free advice whenever you call with any questions about any product.

And when you place an order, we can rush it to you in 48 hours or less. That's the kind of service and support our 10,000 customers have come to expect.

Because we've become a success by giving the best advice for free and selling the best software for less.

To order Microsoft C ($279) or for your free copy of *The C Test*, simply call the toll-free number below:

1-800-421-8006. In Massachusetts, call 1-800-442-8070.

MICROSOFT® C Compiler Version 4.00

MICROSOFT C COMPILER
- Produces fast executables and optimized code including elimination of common sub-expressions. NEW!
- Implements register variables.
- Small, Medium and Large Memory model libraries.
- Compact and HUGE memory model libraries. NEW!
- Can mix models with NEAR, FAR and the new HUGE pointers.
- Library routines implement most of UNIX System V C library.
- Start-up source code to help create ROMable code. NEW!
- Full proposed ANSI C library support (except clock). NEW!
- Link your C routines with Microsoft FORTRAN (version 3.3 or higher), Microsoft Pascal (version 3.3 or higher) or Microsoft Macro Assembler.
- Microsoft Windows support and MS-DOS 3.1 networking support.

MICROSOFT PROGRAM MAINTENANCE UTILITY. NEW!
- Rebuilds your applications after your source files have changed.
- Supports macro definitions and inference rules.

OTHER UTILITIES.
- Library Manager.
- Overlay Linker.
- EXE File Compression Utility.
- EXE File Header Utility.

MICROSOFT CodeView WINDOW-ORIENTED SOURCE-LEVEL DEBUGGER. NEW!
- Watch the values of your local and global variables and expressions as you debug.
- Set conditional breakpoints on variables, expressions or memory; trace and single step.
- Watch CPU registers and flags as you execute.
- Debug using your original source code, the resulting disassembly or both intermingled.

Microsoft C comes with a 30-day money-back guarantee from The Programmer's Shop.

THE PROGRAMMER'S SHOP

The programmer's complete source for software, services and answers.

128 Rockland Street, Hanover, MA 02339 **(617) 826-7531**

Circle **no. 174** on reader service card.

(continued from page 120)

The *for* loop contains the main routine body. The first statement inside the loop prompts the user to enter an abbreviated day name. The rest of the loop contains the exception handling block, starting with the *GET( )* procedure. If a correct day name is entered, the program responds with a brief greeting message and exits from the *for* loop. By contrast, entering an incorrect day name causes a constraint-type error and triggers the exception handling mechanism. The user is given a chance to reenter a correct day name until the loop times out. Notice that one *when* clause in the exception handler deals with constraint errors. The user-defined *Wrong_Day* exception is not used in any *when* clause. Why define it at all? The answer is that it is raised to cause a fatal error and halt the program.

The third approach involves using an alternate method when the first one is plagued by an error. Listing Five, page 104, shows a realistic application for finding the root of a nonlinear function. The main method selected employs the popular and highly efficient Newton's method. This method is vulnerable to functions that have maxima, minima, and saddle points (that is, where the slope is zero) near the root, and so I have used the bisection method (also known as the interval-halving method) as an alternate method. The bisection method is slower but is guaranteed to get a solution on condition that the two supplied guesses form an interval containing the root. The listing shows the use of two nested exception handlers. The first tackles any error generated by Newton's method. This can be a numeric error or divergence error raised by the test for *Max_Iter*. If an error occurs, the program resorts to the bisection method. If this technique encounters any problem (such as an overflow because of a corrupt function), it invokes its own exception handler and triggers the *Fatal_Error* exception. Because there is no handler for the latter exception, the program will then halt.

The fourth approach to exception handling uses a cleanup to alter the error-causing values of one or more variables and then resume program execution. I have modified the application in Listing Five to demonstrate a cleanup operation; the modified code is shown in Listing Six, page 113.

Modula-2 Exceptions

Standard Modula-2 does not have Ada-like exceptions, and so programmers must set up their own error-trapping schemes. Several standard Modula-2 libraries defined by Niklaus Wirth include error-flagging Boolean variables. I will focus on the *MathLib0* library, which provides mathematical functions with no error trapping. Listing Seven, page 114, shows the definition and implementation modules of *SafeLib0*, a subset library of *MathLib0*. This version offers protection against out-of-range arguments for the square root, exponential, and natural logarithm functions.

The definition module *SafeLib0* exports the upper limit for the exponential function. The exported constant can be increased to reflect higher numeric ranges attained by using the 8087 chip in an IBM PC-based Modula-2 implementation. The

(continued from page 123)

definition module also declares the heading of the exported functions. Notice that each mathematical function has one additional argument compared to functions in the standard *MathLib0*. This second argument is *ArgumentERROR*, a Boolean flag to signal that an out-of-range error has occurred. The offending value of the argument is made valid, although the returned function value in the case of argument error is meaningless. I wrote the *SafeLib0* module to track argument errors in an expression containing several mathematical functions. I also included function *GetNext* to return the next error systematically by using an array of Boolean error flags.

Suppose I want to execute the assignment statement shown in Table 1, below. Using *GetNext* allows my application program to pinpoint the particular function and the call (for multiple function calls in an expression).

An alternative method does away with the Boolean out-of-range arguments and instead uses an error stack. Listing Eight, page 115, shows the definition and implementation modules for *SafeLib1*. This version exports a record-typed *ErrorStack* variable. Its structure is composed of a stack height counter and an array of function names. When a mathematical function receives an invalid argument, it pushes its name into the stack and increases the stack height counter.

To use this library, the application program must issue a *ClearErrorStack* before every assignment that involves the mathematical function in question. Following the assignment statement, the library function *InError( )* will indicate if the function arguments were correct. In case of an error, the height counter returns the number of functions that were supplied with invalid arguments. The stack contains the names of these functions. The only drawback of this method over the first one is the case of multiple calls for the same function in an expression. The problem with this method is that the occurrence of the offending function is not apparent.

```
REPEAT
  Pressure := SQRT(Temp,100.,ArgEr[1]) * LN(Volume,22.4,ArgEr[2])
  Current := 0;
  Current := GetNext(Current, 2, Found, ArgEr)
  IF Found THEN
    (* statement to trace errors and either adjust values, or recalculate tTemp or Volume*)
  END;
UNTIL NOT Found;
```

Table 1: *Using* GetNext *in an assignment statement*

Square Matrix Size	Inversion Time (hh:mm:ss.ff)	Comments
10	00:00:00.71	Turbo Pascal
20	00:00:05.16	" "
30	00:00:17.30	" "
50	00:01:19.42	" "
75	00:04:26.61	" "
90	00:07:40.33	" "
100	overflow	" "
140	01:16:33.47	Turbo Extender 20×20-page size, 7 pages
140	01:16:32.32	28×28-page size, 5 pages
140	01:16:33.75	35×35- page size, 4 pages

Table 2: *Matrix-inversion benchmark timings using the 8087 chip*

Large Turbo Pascal Arrays

Turbo Pascal (Versions 1.0 through 3.0) imposes a 64K limit on the data segment, but I have found a product that enables programmers to overcome this barrier. Turbo Extender (a product of TurboPower Software) supplies the user with several interesting alternatives to support large matrices of any type. Each method comes with its own include file. Just how big can these matrices be? The maximum number of columns and rows is 32,767, leading to matrices containing one billion elements! Turbo Extender uses a paging technique that maintains part of the matrix in memory and stores the rest.

The memory schemes used for large matrices are:

1. RAM-based—This method is able to fit matrices in up to 640K of standard memory. The application program must define the size of the RAM-resident portion of the big matrix. The large matrix is made up of a matrix of pages, and so the number of column and row pages must also be defined. This type of matrix is defined at compile time.
2. Disk-based array—This technique stores the matrix in a data file. The application program must specify the same parameters as in the RAM-based version. In addition, the number of pages in RAM must also be defined. Disk-based matrices are defined during program compilation. This alternative applies a virtual-memory method for swapping RAM pages that have been unused for the longest time. The RAM-resident pages' sizes and the number of memory-resident pages determine the speed of accessing the matrix.
3. Virtual arrays—These are very similar to disk-based arrays. The difference is that virtual arrays are dynamically allocated at run time.
4. Expanded-memory arrays—These are similar to virtual arrays, except they reside in the expanded-memory section. The page size is automatically assigned by the library. The application program needs only to specify the number of rows and columns.
5. Sparse arrays—These are dynamically allocated as a linked list. Scanning for sparse matrix elements is done in two phases: the first locates the vicinity of the sought element;

Changing Your Address?

To change your address, attach your address label from the cover of the magazine to this coupon and indicate your new address below.

Name
Address
Address Apt. #
City State
Zip

Mail to: Dr. Dobb's Journal, PO Box 27809,San Diego, CA 92128

STRUCTURED PROGRAMMING
(continued from page 124)

the second pinpoints it.

To use a large matrix of any data type, the application program must first initialize its support. The expanded-memory array needs a second initializing procedure. The actual matrices that are accessed by pointers are dynamically created and optionally initialized by library procedures. Other procedures and functions are available to store and recall matrix elements and clear, delete, save, load, and flush virtual arrays. Error handling is supported via a number of Boolean flags defined in the library.

Listing Nine, page 116, shows a Turbo Pascal program that carries out a matrix-inversion benchmark on a matrix of 140 rows and 140 columns. Notice the initialization and matrix-creation procedures. In addition, you can see the numerous calls for the library routines to store and recall matrix elements.

Table 2, page 124, contains the timing results. I tried three different combinations of page size and number of pages and still obtained the same timings. The table also shows the timings of the 8087-support Turbo Pascal version for smaller matrices. The Turbo Extender manual reports a threefold decrease in speed access for *REALs*. The fact that big matrices can be employed is a welcome feature.

Bibliography

Beidler, J. and Jackowitz, P. *Modula-2*. Boston, Mass.: Prindle, Weber & Schmidt, 1986.

Caverly, P. and Goldstein, P. *Introduction to Ada: A Top-Down Approach for Programmers.* Monterey, Calif.: Brooks/Cole, 1986.

Texel, P. *Introductory Ada Packages for Programming.* Belmont, Calif.: Wadsworth, 1986.

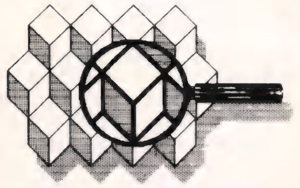
Walker, B. *Modula-2 Programming with Data Structures.* Belmont, Calif.: Wadsworth, 1986.

DDJ

(Listings begin on page 104.)

Vote for your favorite feature/article.
Circle Reader Service **No. 8**.

OF INTEREST



Graphics

Z3D from **Computer Graphics Center** is a new 3-D graphics program for the Macintosh computer. The program displays shadows cast from 3-D objects created by the user and shades these objects. The program has 12 fonts available, which can also be displayed in 3-D. Entire scenes can be viewed from any angle. Z3D is priced at $99. Reader Service No. 16.
Computer Graphics Center Inc.
444 High St.
Palo Alto, CA 94301
(415) 325-3111

The IB-3D1 3D Graphics Package from **SubLOGIC Corp.** contains a set of programs for the creation and real-time animation of 3-D objects. The package includes a custom high-speed assembler/linker, a VIEW program to view and animate 3-D databases, and the Real-Time Animation Language (RTAL) graphic drivers that are also used in several flight-simulator programs. The IB-3D1 3D Graphics Package is available for the IBM PC and compatibles and costs $995. Reader Service No. 17.
SubLOGIC Corp.
713 Edgebrook Dr.
Champaign, IL 61820
(217) 359-8482

NSI Logic has begun shipping the EVC (Enhanced Video Controller)-315, an advanced graphics chip. The EVC-315 integrates five graphics standards, is user programmable, and can operate at a high frequency to achieve high resolutions. The chips are available to OEMs, and the prices begin at $65 per chip for a minimum order of 10,000. Reader Service No. 18.
NSI Logic Inc.
Cedar Hill Business Park
247-B Cedar Hill Rd.
Marlboro, MA 01752
(617) 460-0717

The Inport device interface from **Microsoft Corp.** is for hardware manufacturers who want to integrate graphics input devices into their products. The interface scheme includes a new, 40-pin, custom integrated circuit and a small, compact 9-pin connector. Together the chip and connector reduce the amount of circuit-board and end-bracket space required to provide a graphics input device for personal computers. Reader Service No. 19.
Microsoft Corp.
16011 N.E. 36th Wy.
P.O. Box 97017
Redmond, WA 98073-9717
(206) 882-8080

Definicon Systems has announced MMM, a 32-bit graphics board line for IBM PC-compatible personal computers. MMM combines a million bytes of on-line memory, a million pixels on the screen, and a million instructions per second. The three-board product line comprises two computing engines, DSI-020 and DSI-780, which are based on Motorola's 68020 CPU and 68881 FPU, as well as a graphics processor based on Hitachi's 63484 chip. DSI-020 (16 MHz) costs $1,994, DSI-780 (16 MHz) costs $3,295, the graphics expander board costs $1,495, and the 256-color option for the graphics board costs $400. Reader Service No. 20.
Definicon Systems
31324 Via Colinas, Ste. 108
Westlake Village, CA 91362
(818) 889-1646

A new line of single-slot, IBM PC-compatible, add-in graphics boards are available from **Pronto Computers**. The HR-1200 Series of color graphics boards provide flicker-free graphics on 60-Hz noninterlaced monitors at 1,280 × 1,024 pixels with either 8-bit, 256-color display or 4-bit, 16-color display from a palette of 4,096 colors. The boards also include a 256 × 12 color lookup table, three high-speed digital/analog converters, and 1.5-megabyte memory per screen image. The 1,280 × 1,024-pixel version with 256 simultaneous color display capability is priced at $3,495; the 1,280 × 1,024-pixel board with 16-color capability is $2,795; the 1,024 × 768-pixel board with 256-color capability is $2,895; and the 1,024 × 768-pixel board with 16-color capability is $2,195. Reader Service No. 21.
Pronto Computers Inc.
3730 Skypark Dr.
Torrance, CA 90505
(213) 539-6400

Languages

QuickBASIC 2.0 from **Microsoft Corp.** is a high-performance BASIC compiler that offers high-speed, in-memory compilation and allows users to create structured and modular programs. In addition, its built-in editor and debugger shorten development time, letting programmers write, compile, and debug their programs without having to leave the programming environment. QuickBASIC 2.0 runs on the IBM PC and compatibles and costs $99. Reader Service No. 22.
Microsoft Corp.
16011 N.E. 36th Wy.
P.O. Box 97017
Redmond, WA 98073-9717
(206) 882-8080

UniPress Software has announced UniShell, a Bourne shell script compiler. UniShell analyzes a shell script, translates it into the C language, compiles it using the system C compiler, and then produces an executable program. UniShell programs run faster than shell scripts, can take advantage of the sticky bit and *setuid* for increased efficiency and security, are portable, and have well-structured and readable C code. Prices vary according to computer system. Source code is available for $4,995. Reader Service No. 23.
UniPress Software
2025 Lincoln Hwy.
Edison, NJ 08817
(201) 985-8000

A Modula-2 language system is now available from **Djavaheri Bros.** This Modula-2 runs on the MC68020 processor-based Unix system Altos 3068 Computer System. The price for the product is $495. Reader Service No. 24.
Djavaheri Bros.
697 Saturn Ct.
Foster City, CA 94404
(415) 341-1768

Allen Systems' CP-97 Pascal Cross Compiler for the 8097 16-bit microcontroller on a chip is a complete programming environment oriented around a subset of the Pascal programming

(continued from page 128)

language. The CP (Control Pascal) system is designed to run on IBM PCs and compatibles and includes a screen editor, cross compiler, interpreter, translator, terminal driver, and run-time support software. CP-97 costs $200. Reader Service No. 25.
Allen Systems
2151 Fairfax Rd.
Columbus, OH 43221
(614) 488-7122

The cEnglish Data Base Interface Library from **cLine** has more than 200 functions that provide easy access to the capabilities of the industry-standard file management system C-ISAM. This library allows C applications to be moved easily between MS-DOS, Xenix, and Unix with 100 percent portability across a wide range of systems. The function library includes a broad inventory of tools for screen and memory management, database interfacing, formatting, and utilities. The C functions are accessed by using an English-like preprocessor that creates Lattice C-compatible source code. Reader Service No. 26.
cLine
3550 Camino del Rio North
Ste. 208
San Diego, CA 92108
(619) 281-5593

Cobalt Blue has announced the release of RTC, a Ratfor-to-C translator, in both PC-DOS and VAX-VMS formats. RTC is priced at $450 for PC-DOS and $1,950 for VAX-VMS. Reader Service No. 27.
Cobalt Blue
1683 Milroy, Ste. 101
San Jose, CA 95124
(408) 723-0474

IntelligenceWare's Intelligence/Compiler is an expert-system development environment and knowledge compiler that provides advanced symbolic computing technology on business and industrial computer systems. The Intelligence/Compiler produces code that allows expert systems to be interfaced to conventional languages, applications, and external databases or used on portable computers. Reader Service No. 28.
IntelligenceWare Inc.
9800 S. Sepulveda Blvd.
Ste. 730
Los Angeles, CA 90045
(213) 417-8896

For the IBM PC

International Battery Corp. (IBC) is now marketing lithium replacement batteries for the IBM PC/AT. The AT's internal real-time clock is maintained by an independent battery located on the motherboard when the power is switched off. When this battery fails, the formatting memory of the computer shuts down, requiring total reconfiguration of the hardware. The replacement batteries cost $27.50 each. Reader Service No. 31.
International Battery Corp.
6860 Canby Ave., Ste. 113
Reseda, CA 91335
(818) 609-0516

The **Bubbl-Tec** division of PC/M has announced the PC-1 Bubbl-Board, a magnetic-bubble, mass-storage system for the IBM PC, PC/XT, and PC/AT. The PC-1 provides ½ megabyte of magnetic-bubble memory on a single PC adapter card. The system also provides 512K of nonvolatile mass storage and incorporates intelligent control firmware and circuitry that handles bubble-device formatting and control, interfaces the bubble-memory system to the PC's bus structure, and provides for both soft- and hard-error detection and correction. The 512K version of the PC-1 system is priced at $1,111. Reader Service No. 32.
Bubbl-Tec
6805 Sierra Ct.
Dublin, CA 94568
(415) 829-8700

Sysgen has introduced an internal Winchester subsystem featuring removable hard-disk cartridges for the IBM PC, PC/XT, PC/AT, and compatibles. DuraPak provides PC users with the transportability, security, and unlimited capacity characteristic of removable media. The system is internal, leaving no footprint. The single-drive, 15-megabyte DuraPak system is priced at $1,295, and the 30-megabyte, dual-drive DuraPak is priced at $2,095. Reader Service No. 33.
Sysgen Inc.
47853 Warm Springs Blvd.
Fremont, CA 94539
(415) 490-6770

A new plug-in board for the IBM PC and compatibles, **Microsoft Corp.**'s MACH 10 is designed to improve the PC's ability to run the graphical user interface and multitasking features of Microsoft Windows. The board replaces the 8088 processor chip with a 16-bit 8086 that runs at nearly 10 MHz and uses high-speed cache memory to act as a buffer between the processor and the computer's main memory. MACH 10, including the InPort Mouse and Windows, has a suggested retail price of $549. Reader Service No. 34.
Microsoft Corp.
16011 N.E. 36th Wy.
P.O. Box 97017
Redmond, WA 98073-9717
(206) 882-8080

For the Atari ST

Abacus Software has announced *Introduction to MIDI Programming for the Atari ST* by Len Dorfman and Dennis Young. The book includes the source listings for a comprehensive MIDI editor, driver, and animated player for any of the Casio CZ series synthesizers. The source can be modified for other makes and models. The price of the book is $19.95. Reader Service No. 29.
Abacus Software
2201 Kalamazoo SE
P.O. Box 7211
Grand Rapids, MI 49510
(616) 241-5510

Mark Williams Co. has begun shipping Mark Williams C for the Atari ST. It features a complete implementation of K & R C plus the recent extensions to C implemented under Unix. It includes a shell, utilities, and a full-screen editor and costs $179.95. Reader Service No. 30.
Mark Williams Co.
1430 W. Wrightwood Ave.
Chicago, IL 60614
(312) 472-6659

Networking

A Hayes-compatible 2,400-baud modem, one of the L series, is available from **Leading Edge Hardware Products**. The modem is priced at $289. Reader Service No. 35.
Leading Edge Hardware Products Inc.
225 Turnpike St.
Canton, MA 02021
(617) 828-8150

A linking program from **EKD Computer Sales and Supplies Corp**, pcAnywhere can support up to 19,200 baud, offers conversational and copilot mode, can transfer files bidirectionally, and can be used with 28 popular terminals with customization for

LOGITECH MODULA-2/86 HOLIDAY PACKAGE

$89 Price

- Separate Compilation
- Native Code Generation
- Large Memory Model Support
- Multitasking
- Powerful Debugging Tools
- Comprehensive Module Library
- Available for the PC and the VAX

Use LOGITECH MODULA-2/86 to decrease your overall development cycle and produce more reliable, more maintainable code.

LOGITECH MODULA-2/86 $89

Includes Editor, Run Time System, Linker, 8087 Software Emulation, Binary Coded Decimal (BCD) Module, Logitech's comprehensive library, Utility to generate standard .EXE files. AND more!

LOGITECH MODULA-2/86 with 8087 Support $129

LOGITECH MODULA-2/86 PLUS $189

For machines with 512K of RAM. Increases compilation speed by 50%.

RUN TIME DEBUGGER (Source level!) $69

The ultimate professional's tool! Display source, data, call chain and raw memory. Set break points, variables, pinpoint bugs in your source!

UTILITIES PACKAGE $49

Features a Post-Mortem Debugger (PMD). If your program crashes at run-time the PMD allows you to analyze the status of the program and locate the error. Also includes a Disassembler, Cross Reference Utility, and Version that allows conditional compilation.

LIBRARY SOURCES $99

Source code now available for customization or exemplification.

WINDOW PACKAGE $49

Build windows into your programs. Features virtual screens, color support, overlapping windows and a variety of borders.

MAKE UTILITY $29

Figures out dependencies and automatically selects modules affected by code changes to minimize recompilation and relinking.

CROSS RUN TIME Debugger and ROM Package $199

Still available at an introductory price!

TURBO PASCAL to MODULA-2 TRANSLATOR $49

"Turbo Pascal... is a very good system. But don't make the mistake of trying to use it for large programs."
*Niklaus Wirth**

Our Translator makes it even easier for Turbo users to step up to Modula-2/86. It changes your Turbo source code into Modula-2/86 source, solves all the incompatibilities, and translates the function calls of Turbo into Modula-2/86 procedures. Implements the complete Turbo libraries!

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

30 Day Money Back Guarantee!

To place an order call our special toll free number: 800-231-7717

in California 800-552-8885

$199 Special Holiday Offer

Step up to the power of LOGITECH MODULA-2/86 at a saving of nearly $100 off our usual low prices! We're offering a complete tool set including our compiler with 8087 support (for use with or without an 8087), our Turbo to Modula-2/86 Translator, Run Time Debugger, and Utilities in one holiday package at a special price!

YES I want to step up to LOGITECH MODULA-2/86!

Here's the configuration I'd like:

☐ Special Holiday Package	$199
☐ Logitech Modula-2/86	$89
☐ with 8087 support	$129
☐ Plus Package	$189
☐ Turbo to Modula Translator	$49
☐ Run Time Debugger	$69
☐ Utilities Package	$49
☐ Library Sources	$99
☐ Window Package	$49
☐ Make Utility	$29
☐ ROM Package	$199

Add $6.50 for shipping and handling. Calif. residents add applicable sales tax. Prices valid in U.S. only.

Total Enclosed $________

☐ Visa ☐ MasterCard ☐ Check Enclosed

Card Number Expiration Date

Signature

Name

Address

City State Zip

Phone

LOGITECH

Logitech, Inc.
805 Veterans Blvd.
Redwood City, CA 94063
Tel: 415-365-9852

In Europe:
Logitech SA, Switzerland
Tel: 41-21-879656

In Italy: Tel: 39-2-215-5622

*as reported in Micro Cornucopia, August-September 1985. Turbo Pascal is a registered trademark of Borland International.

Circle **no. 257** on reader service card.

OF INTEREST

(continued from page 130)

other terminal types available. It is priced at $95 for each host machine. The remote computer program ATERM can be copied as many times as required. Reader Service No. 36.
EKD Computer Sales and Supplies Corp.
764 Middle Country Rd.
Selden, NY 11784
(516) 736-0500

PCC/Systems has introduced cc:Mail, a high-end electronic-mail product. Cc:Mail has a file-server-to-file-server message exchange that can be used on individual networks and between networks that are bridged together. All users of local-area networks can exchange text, graphics, and files with any other PC user who is either within or outside the network. Cc:Mail for LANs sells for $995 for a ten-user starter package and is available for a variety of networks. Reader Service No. 37.
PCC/Systems
480 California Ave., Ste. 201
Palo Alto, CA 94306
(415) 321-0430

Miscellaneous

Motorola's Microprocessor Products Group has announced the 25-MHz (40-nanosecond) version of the MC68020 32-bit microprocessor and the 20-MHz (50-nanosecond) version of the MC68881 Floating Point Coprocessor. Prices vary according to quantity and speed. Reader Service No. 38.
Motorola Inc.
Microprocessor Products Group
P.O. Box 3600
Austin, TX 78764
(512) 928-6000

The Disk Defender from **Director Technologies** is a hardware write-protect device for fixed Winchester disks. The product consists of a circuit board, a control box, and a ribbon cable. The circuit board can be installed in either the short or long slot of the IBM PC, PC/XT, and compatible computers. Using full protection, the entire hard disk is readable but not writable. The suggested retail price is $185. Reader Service No. 39.
Director Technologies Inc.
P.O. Box 7067
Evanston, IL 60204
(312) 475-3070

Burton Systems Software is now shipping TLIB, a source-code-revision control system and librarian for PC-DOS and MS-DOS computers. TLIB works with LANs and handles synchronized control of multiple, related source files. A copy of Landon Dyer's MAKE utility is also included. TLIB is not copy-protected and is available for $99.95. Reader Service No. 40.
Burton Systems Software
P.O. Box 4156
Cary, NC 27511-4156
(919) 469-3068

Phoenix Technologies' PC/AT-compatible ROM BIOS for Intel's 80386 is now available. The Phoenix BIOS includes such features as extended central processor diagnostics, full 32-bit memory testing, support for the 80387 numeric coprocessor, and support for processing speeds up to 20 MHz. Reader Service No. 41.
Phoenix Technologies Ltd.
320 Norwood Park S

HAUPPAUGE

Is Getting A Fast Reputation.

HAUPPAUGE started earning a fast reputation with their 87 Math Pak, the combination of an 87 chip and 87 Software Pak that's been accelerating PC math since 1982.

Next came their racy 287 FAST/5, a math coprocessor module with its own 5MHz clock, speeding up PC/AT math by 25%. (Pictured above.)

Now, Hauppauge Unveils the 287 FAST/10...
Our newest math coprocessor for the PC/AT, the 287 FAST/10 moves out at 10MHz—more than doubling the speed of each floating point math operation. The FAST/10 accelerates AutoCad, 1-2-3, Symphony, Turbo Pascal, Framework and more. The FAST/10 also runs in PC/AT compatibles including the Compaq Deskpro 286, Sperry PC/IT and most 286 accelerator boards.

...And the 87 Software Pak Version 6.0
Designed to steal the heart of programmers, the 87 Software Pak supports IBM's BASIC Compiler 1.0 and 2.0, and Microsoft's QuickBASIC, executing math-intensive programs up to 20 times faster! The 87 Software Pak also performs FFT's and Matrix operations. For example, a PC (or PC/XT) with an 87 Chip and 87 Software Pak can perform a 512-point complex FFT in just 1.1 seconds. What's more, a PC/AT with a FAST/10 inverts a 25 by 25 element matrix in under 1 second.

HAUPPAUGE Math Coprocessors

287 FAST/10 10MHz math coprocessor for PC/AT and compatibles...**$469**

287 FAST/8 8MHz math coprocessor for PC/AT and compatibles..**$379**

287 Chip PC/AT math coprocessor—runs at 4MHz in the old model IBM PC/AT and 5.33 MHz in the new model PC/AT..........**$219**

87 Chip Math coprocessor for IBM PC, PC/XT and compatibles..........**$129**

87 Chip Math coprocessor for 8MHz PC compatibles..........**$195**

HAUPPAUGE Math Coprocessor Paks

87 Math Pak V.6.0 87 chip and math coprocessor software support for IBM BASIC Compiler 1.0, 2.0 and Microsoft's QuickBASIC. Plus, Matrix and FFT support, one year of free updates, complete source code and "8087 Applications and Programming"..........**$279**

87 Software Pak V.6.0 Math coprocessor software support as in the 87 Math Pak, but without 87 chip..........**$180**

87 QB PAK Math coprocessor support for microsoft QuickBASIC and IBM BASIC Compiler..........**$ 69**

Recalc+ Math coprocessor support for 1-2-3 version 1A..........**$ 95**

HFT+ Complete Hayes Fourier Transform Package....**$125**

The 287 FAST/10 Doubles Your PC/AT's Math Speed!
Help your PC/AT get a fast reputation with Hauppauge's new 287 FAST/10. Call today, or contact your local computer dealer to learn more about Hauppauge's racy product line. And ask for "87 Q & A," our free booklet on math coprocessors.

Hauppauge Computer Works, Inc.
358 Veterans Memorial Highway, Suite MSI,
Commack, New York, USA 11725
516-360-3827 • TELEX: 262939-HCW

Hauppauge ["Ho-pog"]

OF INTEREST
(continued from page 132)

Norwood, MA 02062
(617) 769-7020

Software Composers has announced Delta Board, a 4½ × 6½-inch circuit board built around the Novix NC4000 single-chip Forth engine. The board provides essentials for immediate application use: stack memory, program RAM, and EPROM with a public-domain Forth interpreter. Delta Board costs $795. Reader Service No. 42.
Software Composers
210 California Ave., Ste. F
Palo Alto, CA 94306
(415) 322-8763

The STD bus-based system from **Devtek Systems** supports program development for 8085, Z80, 8086, and 8088 CPUs. An IBM PC or compatible is used as the system console and for program storage. Programs are downloaded to the STD system where a full complement of debugging utilities can be operated from the PC. The price is $695. Reader Service No. 43.
Devtek Systems
P.O. Box 5224
Lancaster, PA 17601
(717) 560-0652

Viasyn Corp. is now shipping the CompuPro 286/80, a multiuser computer. The CompuPro 286/80 includes an 80-megabyte hard disk with a dedicated 512K cache buffer, a 16-slot S-100R motherboard, a built-in tape backup unit, 800K floppy-disk drive, 768K of main memory, and an 8-MHz 80286 central processor. The computer has a suggested list price of $12,500. Reader Service No. 44.
Viasyn Corp.
26538 Danti Ct.
Hayward, CA 94545-3999
(415) 786-0909

PCMK from **G. W. Glasscock** brings MS-DOS programmers the power of the Unix *make* command. Using information stored in the make file and built-in knowledge, PCMK can automatically construct a program. Features provided by the program are wildcard character expansion, full path-name support, user-defined macros, access to DOS environment via macros, access to internally generated macros, built-in rules defined by users, and direct execution of commands using the environment *Path* variable. The price is $21. Reader Service No. 45.
G. W. Glasscock
15617 S.E. Fairwood Blvd.
Renton, WA 98058
(206) 271-0638

Premier Technologies has announced LiteDrive, a portable internal hard-disk system to enhance the Zenith Data Systems' Z-171 laptop computer. The system adds 27 times the original disk capacity at up to 20 times the access speed. The 10-megabyte system is installed in place of the second floppy and runs on both battery and AC power. Reader Service No. 46.
Premier Technologies Inc.
1890 McGraw Ave.
P.O. Box 16279
Irvine, CA 92714
(714) 261-1184

DDJ

ADVERTISER INDEX

*This advertiser prefers to be contacted directly: see ad for phone number.

ADVERTISING SALES OFFICES

Midwest
Michele Beaty (317) 875-8093

Southeast
Gary George (404) 897-1923

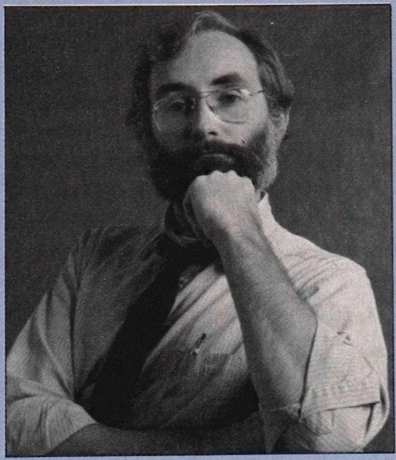
Northeast
Cynthia Zuck (718) 499-9333

Northern California/Northwest
Lisa Boudreau (415) 366-3600

Southern California/AZ/NM/TX
Michael Wiener (415) 366-3600

Advertising Director
Robert Horton (415) 366-3600

SWAINE'S FLAMES



Speaking of languages that you don't speak of, have you programmed in BASIC lately? BASIC, as everyone knows, is not an appropriate language for serious software development, having been expressly created for teaching beginning programmers to program badly. BASIC is slow. BASIC encourages bad programming practices. BASIC is limited to 64K RAM for data. BASIC does not provide a set of development tools. Consequently *DDJ*, although founded to put BASIC in the hands of programmers in 1976, has not wasted space on BASIC recently.

Now there are those who argue, for their own mercenary purposes no doubt, that BASIC is becoming a serious development language. They point to the impressive compiled-code speed measures turned in by the latest implementations. But should we pay any attention to Hal Hardenbergh's claim that he has benchmarked his own HBASIC (née Halgol) for the Atari ST at speeds more comparable to C code than to ST BASIC code?

They say that the control structures of BASIC have evolved to the point that the language actually encourages structured style, that *case* statements and multiline *if-then-else* structures kill the taste for spaghetti code, that with True BASIC and QuickBASIC the language is even getting away from line numbers. They argue that BASIC functions and subroutines are becoming more effective tools for modularizing code, with more flexible parameter-passing techniques and the possibility of defining subroutines to be external or internal. They talk about how the new versions of BASIC offer significant support for function libraries; how QuickBASIC supports separately compiled modules without a linking step; and how Mach 2, a collection of utilities from Microhelp in Marietta, Georgia, embodies a new method for incorporating assembly-language routines into interpreted BASIC programs.

I think these must be the same people who point out that the 64K memory limitation that has kept BASIC in the toy category for years has been eliminated in the modern BASICs and in products such as Mach 2.

But they really show their ignorance when they point out the ease of use of the new BASICs; how they blend characteristics of a compiler and an interpreter; and how the programming environment provided with, say, QuickBASIC is a dream. Don't they know that serious programming languages should not be easy to use? And then they argue that it's time to take a fresh look at BASIC.

Who do these people think they're talking to? If BASIC had become a serious programming language, don't they think you'd already know about it? Why you'd have been reading articles about it in. . . .

Last month I wrote about Stan Kelly-Bootle's yacc (yet another comment compiler), which ignores the code and compiles the comments. My cousin Corbett tells me that he was reminded of SK-B's yacc this week as he read something Niklaus Wirth said: "*Die Grundidee hinter Pascal, wie auch schon bei ALGOL, war, dass man Strukturen einführt, damit die Sätze der Sprache sich später so präsentieren, dass man beim Lesen die Struktur des Programms erkennen kann.*" (*PC Magazin*, vol. 21, no. 22, May 1986.)

Which in real language, Corbett tells me, means something like: "The idea behind Pascal and ALGOL was to invent structures that would allow you to program by writing English *sentences* that implement the program logic." The goal seems eminently right; it's probably what the designer of COBOL would say she had in mind. Corbett maintains that it is obvious that neither of them did it right. He restates the goal this way: In constructing the syntax of a programming language, how close can we get to the ideal of making the efficient implementation of the program in the language and the clear statement of the algorithm in English identical?

Corbett is now engaged in a massive comment compilation project not connected with Stan Kelly-Bootle's scheme. He argues that if we knew just how programmers wanted to express their algorithms in English, we would have a model for the syntax of the ideal programming language, the one that makes the English description and the code identical. But the kinds of sentences we want to write describing the logic of programs, he claims, are just the ones we are already writing—called comments—only spelled better. Corbett plans to do a linguistic analysis of programmers' comments—along the lines of the textual analyses used to establish authorship of manuscripts—to determine the kinds of language constructs we really need in the perfect programming language.

Michael Swaine

Michael Swaine

Why more than half a million people are using Turbo Pascal

Because Turbo Pascal is faster than any other Pascal compiler—it's become the industry standard. And because Turbo Pascal is backed by a complete range of "tool boxes" that give you most of the programming tools you'll ever need.

- Turbo Pascal® 3.0 with 8087 and BCD
- Turbo Database Toolbox™
- Turbo Graphix Toolbox®
- Turbo Tutor® 2.0
- Turbo Editor Toolbox™
- Turbo GameWorks®

The Turbo Pascal family is never static, but is continuously expanding, with new products like Turbo Editor Toolbox and Turbo Gameworks.

> "Language deal of the century . . . Turbo Pascal
> *Jeff Duntemann, PC Magazine*
>
> Turbo Pascal has got to be the best value in languages on the market today
> *Jerry Pournelle, BYTE Magazine*
>
> This compiler, produced by Borland International, is one of the best programming tools presently available for the PC
> *Michael Covington, PC Tech Journal*"

The secret of software success is not merely low price, but top quality allied with complete documentation, like our 400-page reference manual.

All of which are some of the reasons why Turbo Pascal is clearly the leader, and the recipient of awards like *PC Week's* "Product of the Year" and *PC Magazine's* "Award for Technical Excellence."

NEW! TNT* A DYNAMITE DEAL!
**Turbo 'N Tutor*
Turbo Pascal *and* Turbo Tutor for only $125.00. You save nearly $15.00, and have everything you need to get up and go with Turbo Pascal!

A whole family of tools

Success breeds success, so the Turbo Pascal family has flourished. Your choices now include:

☐ **Turbo Pascal 3.0** combines the fastest Pascal compiler with an integrated development environment, *and* 8087 math co-processor support *and* Binary Coded Decimals.

☐ **Turbo Database Toolbox** a perfect complement to Turbo Pascal. It includes a complete library of Pascal procedures that allows you to search and sort data and build powerful database applications.

☐ **Turbo Graphix Toolbox** includes a library of graphics routines for Turbo Pascal programs. Lets even beginning programmers create high-resolution graphics with an IBM,® Hercules,™ or compatible graphics adapter. Does complex business graphics, easy windowing, and stores screen images to memory.

☐ **Turbo Tutor 2.0** teaches you step-by-step how to use Turbo Pascal, with commented source code for all program examples on diskette.

An offer you can't refuse

You can save $119.75 when you choose the Turbo Jumbo Pack. Six different Turbo Pascal products for only $299.95.

You get Turbo Pascal 3.0 *and* Turbo Editor Toolbox *and* Turbo Tutor 2.0 *and* Turbo Graphix Toolbox *and* Turbo GameWorks *and* Turbo Database Toolbox!

So act now—rush to your dealer or call us today!

NEW! Amazing value! Turbo Editor Toolbox includes MicroStar, a full-blown editor that also does windows! You get ready-to-compile source code and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs. Also includes MicroStar, a complete editor with full windowing capabilities. (You could pay $100.00 or more for a program like MicroStar, but you get it free as part of our Turbo Editor Toolbox.) You can also use Turbo Editor (which of course integrates with Turbo Lightning®) to build your own word processor!

NEW! Turbo GameWorks gives you the games you can play, write, rewrite, bend and amend! Turbo GameWorks reveals the secrets and the strategies of game design. You're given source code, a 200-page manual, and the insight needed to write and customize your own irresistible games.

Also includes ready-to-play Chess, Bridge, and Go-Moku—an ancient Japanese game that can divert you from reality for hours on end.

YES! I want the best!

To order by phone, or for a dealer nearest you, **call (800) 255-8008**
in CA call (800) 742-1133

Copies	Product	Price	Totals
___	Turbo Pascal 3.0 w/8087 & BCD**	$ 99.95	$ ______
___	Turbo Pascal for CP/M-80	69.95	$ ______
___	Turbo Database Toolbox	69.95	$ ______
___	Turbo Graphix Toolbox*	69.95	$ ______
___	Turbo Tutor 2.0	39.95	$ ______
___	Turbo Editor Toolbox*	69.95	$ ______
___	Turbo GameWorks*	69.95	$ ______
___	Turbo Pascal & Turbo Tutor	125.00	$ ______
___	Turbo Jumbo Pack*	299.95	$ ______
	Outside USA add $10 per copy CA and MA res. add sales tax		$ ______
	Amount enclosed		$ ______

Prices include shipping to all US cities.

Carefully describe your computer system:

Mine is: __ 8-bit __ 16-bit

I use: __ PC-DOS __ MS-DOS __ CP/M-80 __ CP/M-86

My computer's name and model is: ______________________

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Payment: VISA MC Money Order Check

Credit card expiration date ___/___

Card # |_|_|_|_|_|_|_|_|_|_|

TP11

Name: ______________________

Shipping Address: ______________________

City: ______________________

State: ____________ Zip: ____________

Telephone: ______________________

CODs and purchase orders WILL NOT be accepted by Borland. Outside USA make payment by bank draft, payable in US dollars drawn on a US bank.

NOT COPY PROTECTED
60-DAY MONEY-BACK GUARANTEE

If within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department and we will gladly arrange a refund.

All prices are suggested list prices and are subject to change without notice.

Minimum System Requirements:
Turbo GameWorks, Turbo Graphix Toolbox, Turbo Tutor, & Turbo Editor Toolbox—192K. All other products, 128K.

*IBM PC, PCjr, AT, XT, and true compatibles.

**16-bit only.

BORLAND INTERNATIONAL
4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Vive la différence

Circle **no. 161** on reader service card.